# *Entwurf und Realisierung eines mobilen autonomen Systems variabler Intelligenz (MauSI$^{(R)}$)*

*Dissertation*

*zur Erlangung des akademischen Grades*
*Doktoringenieur*

*der Fakultät für Informatik und Automatisierung der*
*Technischen Universität Ilmenau*

vorgelegt von:

***Dipl.-Ing. Jens Altenburg***
geb. am 26. September 1964 in Jena

Berichterstatter:

Uni.-Prof. Dr.-Ing. habil. J. Wernstedt, TU Ilmenau
Prof. Dr.-Ing. K.-D. Morgeneier, FH Jena
Dr.-Ing. H. Wächter, Desotron GmbH

Tag der Einreichung: 11.12.2003

Tag der öffentlichen wissenschaftlichen Aussprache: 05.07.2004

Verfahren Nr.: IA 125

# Vorwort

Wann zum ersten Mal künstliche Helfer zur Erledigung schwerer, langweiliger oder gefährlicher Arbeiten erdacht und gebaut worden sind, läßt sich wohl kaum genau bestimmen. Mit einer gewissen Berechtigung können mechanische Rechenmaschinen bzw. Automaten zu den Urahnen moderner Robotik gezählt werden. Erste Roboter im eigentlichen Sinn, d.h. mehr oder minder intelligente und zumindest teilweise autonome Systeme sind erst im 18. Jahrhundert als mechanische Schreibautomaten oder Nachbildung einer mechanischen Ente bekannt geworden. Ein ebenfalls mechanischer Schachspieler stellte sich schnell als Schwindel heraus, offenbar ist Scharlatanerie wohl noch älter als Robotertechnik.

Wirklich ernstzunehmende Konstruktionen datieren erst aus der ersten Hälfte des 20. Jahrhunderts. Diesen zum Teil noch stark experimentellen Geräten ist ihr potentieller Ansatz zu Produktion und Rationalisierung deutlich anzusehen. Die Weiterentwicklungen dieser Handhabeautomaten leisten heute als Industrieroboter Erstaunliches. Dennoch können solche Systeme infolge ihrer hohen Kosten erst in der Großserienproduktion ihre Fähigkeiten voll entfalten.

Erst der extreme Preisverfall und die enorme Leistungssteigerung informationstechnischer Bauelemente wie Prozessoren, Speicher und Peripheriemodule schafft die Grundlage für (zumindest teil-) autonome Systeme, die innerhalb bestimmter Grenzen selbstständig Probleme erkennen und einer Lösung zuführen können. Mit der vorliegenden Arbeit wird der Versuch unternommen mit minimalem Kostenaufwand einen variablen anpassungsfähigen mobilen Roboter zu definieren, welcher als Bestandteil eines übergeordneten (globalen) Systems Teilaufgaben zugewiesen bekommt und diese auf lokaler Ebene bewältigt.

Besonderer Dank für die Begleitung und Unterstützung dieser Arbeit gilt von Seiten der TU Ilmenau Herrn Prof. Dr. Wernstedt und Herrn Dr. Rauschenbach. Der recht hohe Anteil mechanischer Konstruktionen ist in vielfältiger Art und Weise von den Firmen CT-Videotechnik GmbH, Rothenschirmbach, Mikrostep GmbH, Sömmerda und nicht zu letzt von Herrn Dipl.-Ing. Helmut Altenburg unterstützt und gefördert worden.

Sömmerda, den 12. 10. 2003 Jens Altenburg

# Inhaltsverzeichnis

# 1 Einleitung

## 1.1 Ausgangssituation

Der Zwang zur ständigen Rationalisierung von Produktions- oder Herstellungsverfahren führt zum zunehmenden Einsatz von Industrierobotern. Diese Systeme ermöglichen die hochproduktive Herstellung von Massenbedarfsartikeln. Optimierte Programmstrategien, schnelle Rechentechnik und neue Antriebssysteme erschließen immer neue Anwendungsfelder.

Zunehmend werden jedoch auch Automatisierungssysteme in Bereichen verlangt, die außerordentlich flexible Einsatzstrategien erfordern. Dabei sei an "Dienstleistungsroboter", "mobile Informationsterminals" oder audiovisuelle "Inspektionssysteme" gedacht. Neben der großen Varianz der vorstellbaren Einsatzbedingungen setzt der beschränkte Kostenrahmen solchen Systemen enge Grenzen.

Während in der Industrieautomation die Kosten hochproduktiver Industrieroboter (je nach Anwendung ist mit sechsstelligen Investitionssummen zu rechnen) auf die Stückzahl umgelegt, auch relativ hohe Investitionssummen rechtfertigen, sind in den erwähnten Bereichen zu hohe Kosten ein k.o.-Kriterium für die Anwendung. Hinzu kommt auch eine gewisse "Akzeptanz-Schwelle". Der Einsatz von Servicesystemen im Gesundheitsbereich, z.B. zum Transport von Essensportionen oder Bettwäsche, stößt nicht auf jedermanns Zustimmung. Doch ist auch hier durch den enormen Kostendruck in absehbarer Zeit ein Umdenken zu erwarten. Es ist sicher in jedem Fall sinnvoll, qualifiziertem Fachpersonal zur Erfüllung seiner eigentlichen Aufgabe Servicesysteme zu Transportaufgaben bereitzustellen, damit diese, um beim geschilderten Beispiel zu bleiben, dem Patienten mehr Zeit widmen können. Eine gewisse Gratwanderung ist der Einsatz von Servicesystemen dennoch. Zu leicht kann es zur "Entmenschlichung" aus Gründen reinen Kostenmanagments in bestimmten Bereichen kommen; doch ist dies mit Sicherheit nicht hauptsächlich die Schuld der Roboter.

Noch ist jedoch die technische Entwicklung vom breiten Einsatz mobiler autonomer Systeme weit entfernt. Trotz zum Teil verblüffender Fortschritte in der Computertechnik sind bestimmte Fragen in der Führung und Kontrolle autonomer Systeme ungelöst.

Zwei wesentliche Verfahren "konkurrieren" miteinander:

> ***Weltmodell*** - innerhalb des autonomen Systems wird aus den Sensordaten ein Abbild der Umgebung modelliert. Alle Entscheidungsprozesse werden danach mit Hilfe dieses Modells gewonnen und repräsentieren damit das Außenverhalten des Systems.

> ***Verhaltensmodell*** - Sensordaten werden hierarchisch priorisiert mit fertig programmierten Verhaltensweisen verknüpft; das autonome System reagiert ausschließlich (und unmittelbar) auf seine Sensoren.

Der signifikante Unterschied beider Verfahren liegt in der prognostizierten Rechenleistung, die zur Steuerung einer autonomen Einheit notwendig ist.
Ein idealisiertes Weltmodell bedeutet eine komplette Abbildung der Umgebung des autonomen Systems mit Hilfe seiner Sensorik auf entsprechende mathematische Verfahren, aus denen notwendige Entscheidungen abgeleitet werden sollen. In welcher Form ein solches Modell gefunden werden kann, scheint noch weitgehend unbekannt zu sein. Mit einiger Sicherheit ist lediglich ein erheblicher Aufwand an Rechentechnik absehbar.
Einen pragmatischen Lösungsansatz liefert das Verhaltensmodell. Ausgehend von frühen Versuchen aus der Kybernetik (Wiener, 1948, 1961; Grey 1950, 1951) wurden am MIT von Brooks (1986) weitergehende Untersuchungen zur Steuerung autonomer Roboter unternommen [1], [31]. Die Weiterentwicklung der zur technischen Umsetzung erforderlichen Bauelementebasis (energiesparende Mikrocontroller, Miniaturmotoren, neue Akku-Technologien) boten bessere Chancen, die Intelligenz in das autonome System zu verlagern. Gänzlich ist die Verlagerung intelligenter Algorithmen dennoch nicht gelöst. Offenbar existiert nach wie vor ein Problem zwischen den begrenzten Ressourcen autonomer mobiler Einheiten und der im Minimum nötigen Rechenleistung für intelligente Verhaltens- oder Systemsteuerung. Bekannte Verfahren zur intelligenten Steuerung erfordern einen zum Teil beachtlichen Rechenaufwand, dazu basieren sie häufig auf Computerplattformen (PC-Technik, Betriebsysteme Windows 95/98/NT, Linux etc.), die sich nicht oder nur sehr ungenügend nach "unten hin" abrüsten lassen. In der Öffentlichkeit bekannt gewordene und stark beachtete Intelligenzleistungen von Rechnern bzw. Rechnersystemen, wie z.B. der Sieg von *Deep Blue II* über den amtierenden Schachweltmeister, basieren auf vergleichsweise riesigen Anlagen. Solche Systeme sind schwerlich in mobile Einheiten integrierbar.
Inwiefern sich mit Hilfe von ausschließlich verhaltensmodellbasierten Systemen "richtig" intelligente Systeme aufbauen lassen, ist noch nicht abschließend beantwortet. Der auf den ersten Blick verblüffend einfache Ansatz birgt bei der Implementierung in komplexere Systeme deutliche Schwächen [11].

## 1.2 Aufgabenstellung der Dissertation

Die Zielstellung dieser Arbeit besteht darin, ein autonomes mobiles System zu definieren, zu entwickeln und als Prototypen zu testen, welches in der Lage ist, sich im zweidimensionalen Raum (R'2) zu bewegen und dabei nach Vorgabe frei wählbare Zielpunkte anzusteuern. Dabei wird gefordert, daß nur geringstmögliche Eingriffe durch übergeodnete Instanzen, Bediener oder Operator bzw. rechnerbasierte künstliche Intelligenz notwendig, sind. Ein weiteres Ziel der Arbeit besteht darin, die gewonnenen Ergebnisse in einen Demonstrationsaufbau einfließen zu lassen und damit ihre Praxiseignung zu beweisen. Der Einsatz kostensensitiver, industriell umsetzbarer Algorithmen bzw. Implementationen wird als wesentlich empfunden. Anzustreben ist die Kopplung der mobilen autonomen Einheiten an ein Führungssystem, vorzugsweise als Rechnersystem mit übergeordneter Planungs- und Entscheidungskapazität, so daß über interaktive Eigenschaften von Planungseinheit (Missionsmanagment) und lokaler Intelligenz (mobile Einheit) flexible Lösungen zur Führung von mehr als einer mobilen Einheit entstehen.
Offenbar sind mehr oder minder alle industriell genutzten Transport- oder Förderanlagen, diese kommen von ihrer Aufgabenstellung dem geforderten System am nächsten, auf einen nicht zu unterschätzenden personellen Zusatzaufwand, der einen breiten Einsatz unter realen Kostenrelationen verhindert. Alle diese Systeme benutzen Orientierungshilfen, meist in Form von Leitlinien [9], die z.B. optisch oder magnetisch die mobile Einheit auf ihren Weg leiten. Für erhöhte Zuverlässigkeit werden zum Teil auch spurgeführte Systeme verwendet.
Die Problematik zeitweiser (kurzzeitig) autonomer Systeme, die sich in nur teilweise bekannter Umgebung zielgerichtet orientieren müssen, ist durchaus bekannt, aber mangels leistungsfähiger und bezahlbarer Lösungen in der Breite noch nicht einsetzbar.

Aus dieser kurz umrissenen Ausgangslage kristallisiert sich ein mehrstufiger Forderungskatalog für die Entwicklungsabfolge heraus:

1. Analyse bekannter Steuermodelle (*Weltmodell*, *Verhaltensmodell*) sowie Erweiterung und Verbesserung erreichter Lösungsansätze (*virtueller Zielpunkt*)

2. Ergänzen der Verhaltenstrategien durch Implementierung eines internen Beobachters zur laufenden Selbstlokalisierung von einem bekannten Ausgangspunkt (*inertialer Beobachter*) und Gefahrenabwehr

3. Einführung eines *Verhaltensobjektes* als zentraler Bestandteil verknüpfter Strukturen aus Daten (Koordinaten zur Lageerkennung) und implementierten Programmteilen

(*Verhaltensmodulen*). Definieren von grundlegenden Verhaltensobjekten zur intelligenten Führung autonomer mobiler Systeme

4. Bereitstellen von Algorithmen zur Bewertung von Kartendaten und Sensorinformationen des unmittelbar zugänglichen Arbeitsraumes. Darauf basierende Pfadneuplanungen zur alternativen Wegneuplanung bzw. Kollosionsvermeidung. Sicherstellen der Anpassungsfähigkeit des Systems an verschieden komplexe Forderungen durch Erweiterung der Verhaltensobjekte zu *Objektlisten*

5. Realisierung von drahtlosen Kommunikationskanälen zur lokalen Datenübertragung (zwischen autonomen Einheiten) und zur Kontaktaufnahme mit der übergeordneten Instanz (globale Intelligenz)

6. Definition und Aufbau einer Testplattform (Roboterfahrzeug). Ausrüstung des Fahrzeuges mit Standardsensorik (Infrarot-Lichtschranken) und Regelalgorithmen zur Motorsteuerung, sowie Integration komplexer Sensorsysteme (Beschleunigungssensoren, bildgebende Sensoren).

7. Implementierung eines leistungsfähigen Softwaresystems. Schwerpunkte bei der Realisierung bilden Multitaskingfähigkeiten, Unabhängigkeit von kommerziellen Systemen, d.h. keinerlei Zahlung von Lizenzgebühren (*royalty free*) sowie Portierbarkeit (Implementierung auf anderen Hardwareplattformen).

8. Bereitstellung von Algorithmen für die Sensordatenanalyse. Neben einfachen Verfahren (Dateninterpolation bzw. Störminimierungen) bilden Experimente zur Bildverarbeitung auf der lokalen Einheit einen Schwerpunkt der Untersuchungen.

9. Untersuchung konventioneller Pfadplanungen (Kartendateninterpretation, Potentialfeld gestützte Planung, etc.) im Vergleich zu symbolischen Pfadplanungen bzw. Suchstrategien.

10. Ableitung der Steuer- und Regelalgorithmen für andere Fahrzeugplattformen. Prüfung der Ergebnisse in (potentiell) industriellen Anwendungen.

Insbesondere auf die Realisierung der gefundenen Ergebnisse wird im Hinblick auf industrielle Einsatzfähigkeit großer Wert gelegt.

Da die autonomen Einheiten als potentielle Verlustgeräte eingestuft sein können, liegt ein weitere Schwerpunkt auf der Frage der Kostenminimierung bei gleichzeitiger größtmöglicher Sicherung gewonnener Daten (Weginformationen) der mobilen Einheit.

# 2 Steuerverfahren und Entscheidungsmodelle

## 2.1 Antriebs- und Steuerkonzepte bekannter Robotersysteme

### 2.1.1 Radantriebe

Die Kurzvorstellung wichtiger Antriebs- und Steuerverfahren liefert einen Überblick zum derzeitigen Stand auf dem Gebiet mobiler autonomer Robotiksysteme.
Das Mittel der Wahl sind in vielen Fällen Radantriebe. Für industrielle Transportsysteme dominieren vierrädrige Plattformen. Dreiradkonstruktionen bzw. Modifikationen des *differential drive* sind weniger zu finden. Der Grund liegt in der besseren mechanischen Stabilität (geringe Kippgefahr) von Vierradantrieben.
Die meisten üblichen Radantriebe fungieren als nicht-holonome Plattformen. Nicht-holonom bedeutet in der Praxis, daß die Roboter nicht einfach als Punkt- bzw. Flächenmodell für die Navigation betrachtet werden können.

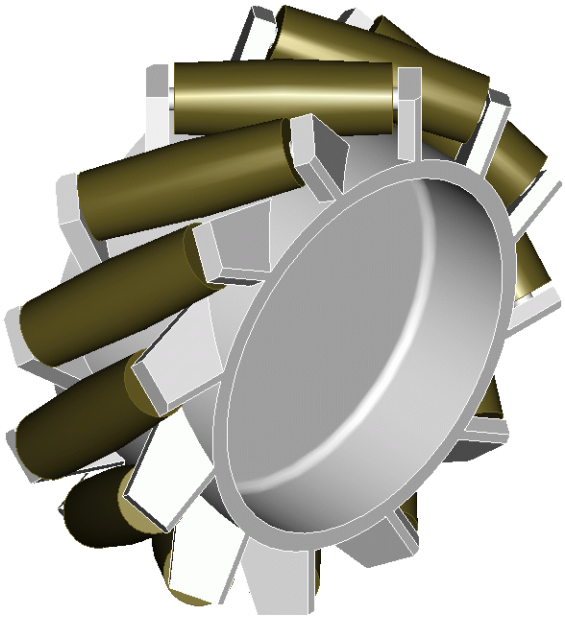


Bild 2.1 Aufbau eines Mecanum-Rades

Ein vierrädriger Transportroboter kann, wenn er z.B. in eine Ladebucht ansteuert, die seine Außenabmaße nicht wesentlich übersteigt, nur mittels komplizierter Rangierbewegungen seitlich verschoben werden. Um eine Ladungsübernahme so einfach wie möglich

zu gestalten muß ein solcher Roboter rückwärts andocken, und kann dann mit minimalem Abstand auch von dort aus beladen werden.
Hat ein solches System mehrere Entladestationen, ist die Reihenfolge des Beladens ausschlaggebend für die anzusteuernde (Entlade-)Route. Kann eine Entladestation nicht bedient werden, stockt u.U. der gesamte Versorgungsprozeß. Sehr viel günstiger wären seitliche Be- bzw. Entlademöglichkeiten. Dies ließe sich zwar mit langen "Laderampen" lösen, aber diese brauchen Platz, der in Lagerhallen oder Fabrikgebäuden knapp und teuer ist.
Eine relativ exotische Lösung für Radantriebe ist das Mecanum-Rad. Wie in der Abbildung zu sehen, besteht das Mecanum-Rad aus einer speziellen Mechanikkonstruktion, bei der auf einem Innenrad weitere Räder, deren Achsen versetzt zur Radachse stehen, angebracht sind.
Ein Roboter, der mit vier derartigen Räder angetrieben wird, ist als holonomes System aufzufassen. Die meisten anderen Roboter, typischerweise mit einem *differential drive* versehen, sind nicht-holonom.
"Holonomische Systeme sind durch den Fakt charakterisiert, daß die Rückkehr aller Aktoren zu ihrer Ausgangsposition den gesamten Roboter in seine Ausgangsposition zurücksetzt. Nicht-holonomische Bewegung anderseits beschreibt Systeme, die nicht zu Ihrer Ausgangsposition zurückfinden, wenn alle Aktoren entsprechend zurückgesetzt werde. In einem nicht-holonomischen System ist der Status des ganzen Systems pfadabhängig." [8].
Ein holonomer Roboter mit Mecanum-Rädern kann ohne rotatorische Bewegungskomponente jeden Punkt im zweidimensionalen Raum (R'2) erreichen.

### 2.1.2 Schreitroboter

Neben den Mecanum-Rad besitzen auch Schreitroboter einen gewissen "Exotenbonus". Untersuchungen zu Schreitsystemen sind jedoch so selten nicht. Auch zur Anzahl der benutzten Beine gibt es viele Publikationen. Angefangen von einbeinigen Systemen reicht die Palette über mehr oder minder menschenähnliche "Zweibeinkonstruktionen" (Bipeds) zu einer sehr großen Varianz in der Ausnutzung insektenähnlicher Bewegungsmuster.
Neben der besseren Stabilität sechsbeiniger Plattformen, sind die biologischen Vorbilder mit teilweise verblüffenden Steuer- und Regelmechanismen ausgestattet. Offenbar sind Rechenleistung und Übertragungskapazität bei Insekten kostbare Güter die nur begrenzt zur Verfügung stehen. Ohne in Details zu gehen, "programmiert" eine Gottesanbeterin ihre Fangbeine mit bestimmten Bewegungsmustern bevor sie ihre Beute packt. Die

Bewegung der Fangbeine erfolgt so schnell, daß eine Korrektur im Bewegungablauf nicht mehr möglich ist. Weder kann das primitive Gehirn des Insekts neue Bewegungsabläufe bestimmen, noch können diese über die relativ langsamen Nerven an die Aktoren übermittelt werden.
Ein ähnliches, wenn auch nicht so martialisches Problem besteht in der gerichteten Koordination der sechs Beine während des Laufens. Sieht man die recht ungelenk wirkenden Versuche, dies elektromechanisch nachzubilden, wird der innewohnende Steueraufwand erahnbar.

Bild 2.2 Hexapode RHex der McGill University Quebec [30]

Einige Schabenarten lösen das Problem der koordinierten Bewegung von Beinen derart, daß sie überhaupt nicht koordinieren, jedenfalls nicht alle Beine. Die Parole "Marsch. marsch,.." erhält der Einfachheit halber nur das erste Beinpaar. Die folgenden Beinpaare sind mit Neuronen untereinander verbunden und erhalten ihre Steuersignale erst dann, wenn das vorausgegangene Paar seine Bewegung eingeleitet und Platz für die nächsten Beine geschaffen hat.
Ein solcher Mechanismus ist natürlich einfach beschrieben, ob er wirklich funktioniert, steht auf einem anderen Blatt.

Die Roboterkonstruktion RHex setzt mit ihrem recht eigentümlichen Aufbau Maßstäbe, was die Überwindung auch widrigster Hindernisse betrifft. Unter [30] sind eine Reihe von Videoclips zu sehen, die diese Fähigkeit augenscheinlich demonstrieren.

### 2.1.3 Roboterkolonien

Die Anzahl der möglichen Sonderbauformen und Spezialantriebe läßt sich erheblich erweitern. Außer den "erdgebundenen" Systemen sind auch schwimmende, tauchende oder fliegende Roboter bekannt.

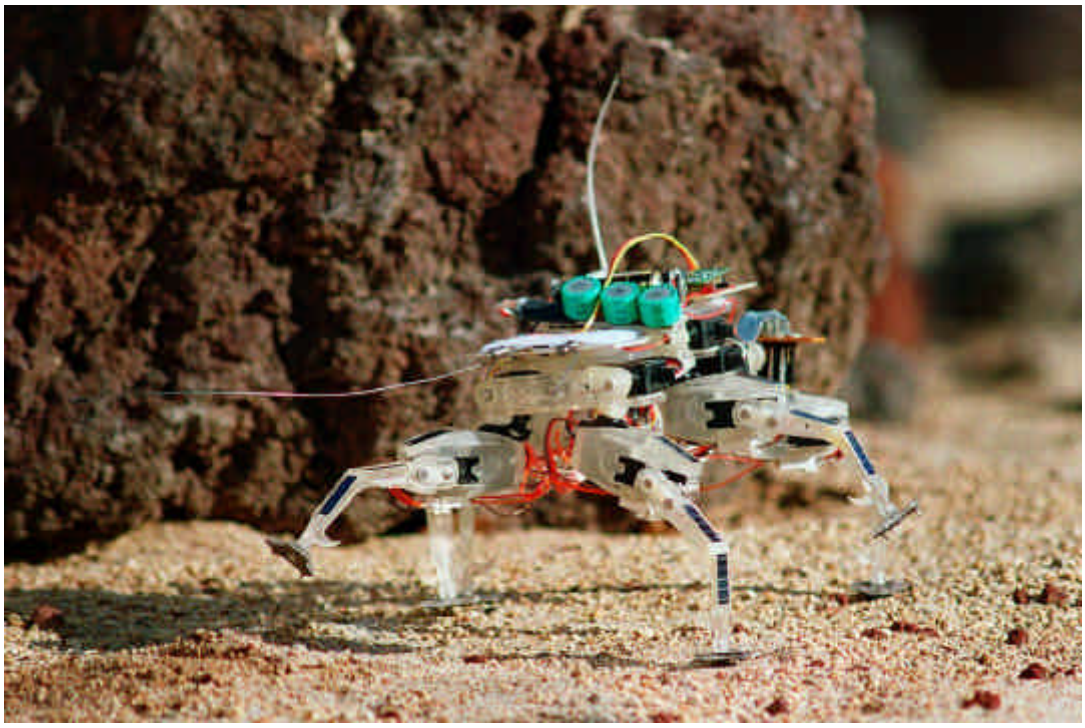


Bild 2.3 Spider-Bots für Geländeerkundung [31]

Neben diesen Einzelsystemen gibt es jedoch auch versuchsweise Ansätze, mehrere Roboter mehr oder minder im Team arbeiten zu lassen. Die Vorteile liegen auf der Hand. Roboter sollen möglichst universell verwendbar sein. Universalität hat in der Technik aber ihren Preis. Außer den reinen Fertigungskosten, die bei möglichst vielfältig einsetzbaren Systemen anfallen, zieht die universelle Verwendbarkeit oft eine Reihe von Kompromissen nach sich, die beinahe immer zu Lasten einzelner Teilleistungen gehen.
Die Entwicklung mehrerer auf bestimmte Teilgebiete spezialisierter Roboter scheint hier Abhilfe zu schaffen. Der einzelne Roboter kann zwar nicht mehr alles, in Kombination

mit anderen ist dieser Nachteil offenbar behebbar. Voraussetzung hierzu ist die Kordination der verschiedenen "Roboterfähigkeiten" untereinander.
Als weiterer Pluspunkt schlägt die geringe Anfälligkeit gegen ein gleichzeitiges Komplettversagen mehrerer Roboter zu Buche.
Am Jet Propolsion Laboratory sind "Spider-Bots" als universelle krabbelnde Planetenerkunder vorgestellt worden [31]. Der Entwickler, Robert Hogg, verspricht sich von seinen Systemen sichere Erkundungsmissionen, Kommunikation von Robotern untereinander mit wechselseitiger Abstimmung und, sollten Systeme zerstört werden, kann die Mission, zumindest teilweise, von den verbleibenden Robotern erfüllt werden.
Aus der Abbildung 2.3 wird der low-cost Ansatz dieser Entwicklung sichtbar. Dennoch sollen die Roboter mit diversen Sensoren, Kameras und Funkmodulen ausgerüstet werden können.
Auch ein Bestücken der Roboter mit diversen Werkzeugen soll möglich sein.

## 2.2 Entscheidungsmodelle

### 2.2.1 Weltmodell

In der Literatur (u.a. [1], [2], [3]) werden zwei grundlegende Steuerverfahren beschrieben. Das erste Verfahren bedient sich eines "Weltmodells" zur Koordinierung der Roboterfunktionen. Dieses Modell bildet die zentrale Datenbasis aller Entscheidungsprozesse der Robotersoftware. In dieses Modell gehen sämtliche Sensorinformationen der mobilen Einheit ein.
Infolge der Vielzahl der notwendigen Sensordaten, die sich sowohl aus externen Informationen, Lagedaten, Navigationsinformationen, Umweltsignalen etc. als auch aus internen Zustandsberichten, wie Ladezustand der Akkumulatoren o.ä, zusammensetzen, ist davon auszugehen, daß der vollständige Entwurf eines solchen Modells aufwendig ist.

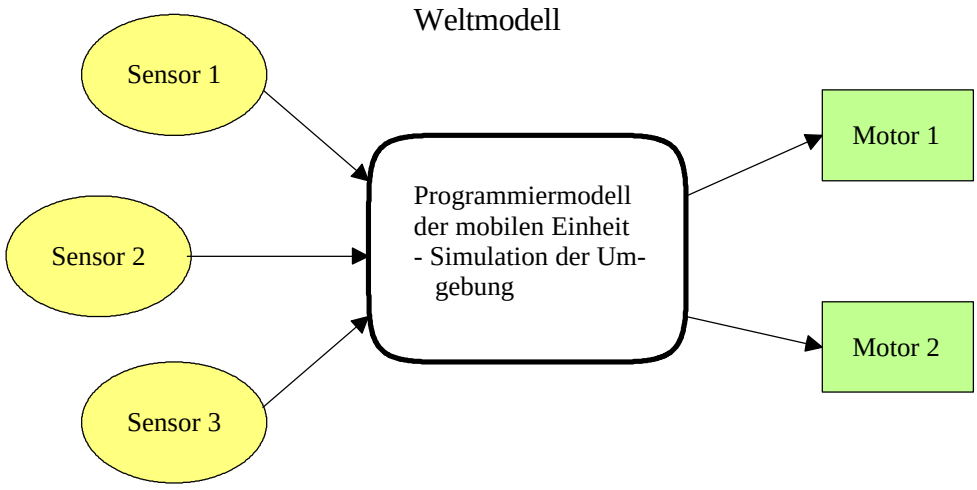



Bild 2.4 Struktur eines einfachen Weltmodells

Die Abbildung 2.4 illustriert den Aufbau eines einfaches Weltmodells. Eine Anzahl Sensoren aktualisiert die Datenbasis des Systems. Bei bestimmten Sensoren, Abstandsmeldern, Lagedetektoren müssen diese Berechnungen in Echtzeit erfolgen.
Aus den Sensorinformationen werden mit Hilfe des Programmiermodells die Steuerinformationen für die Antriebsmotoren gewonnen. In [3] wird ein Weltmodell zur Auswahl eines zurückzulegenden Fahrweges beschrieben.
Hinsichtlich der Zielstellung der Arbeit liegt der Schwerpunkt der dort geschilderten Untersuchungen auf der Definition eines allgemein nutzbaren Programmieransatzes. Im

Endeffekt soll das Problem der Wegplanung bzw. Auswahl auf der Ebene der mobilen Einheit erkannt und gelöst werden.
Das Weltmodell strukturiert diese Auswahl mit Hilfe eines mehrschrittigen Vorgehens, das sich wie folgt beschreiben läßt:

Schritt 1: Auswertung und Analyse der Sensordaten zum Ermitteln grundlegenden topologischen Umweltwissens; nach Möglichkeit Aufbereitung der Daten zu einer Umgebungskarte

Schritt 2: Auswahl einer durchgängig befahrbaren Trasse (Trajektorie) zwischen Start und Zielpunkt

Schritt 3: Anpassung dieser Trajektorie an die geometrischen, kinematischen und kinetischen Verhältnisse des realen Fahrzeuges mit anschließender Umsetzung in Fahrbefehle

Die Punkte 1 und 2 sind primär unabhängig von der Hardware des realen mobilen Systems. Lediglich in Schritt 3 werden die Eigenschaften des realen Systems berücksichtigt. Im Grunde entspricht dieses Vorgehen weitgehend dem eines Menschen, der eine solche Aufgabe zu lösen hätte.
Während Schritt 3 soll sich die mobile Einheit in Bewegung befinden. Mit einiger Berechtigung können diese Schritte zu zwei Phasen, der Phase der Wegplanung und der Phase der Bewegung (Fahrt) zusammengefaßt werden. Darüber hinaus ist klar, daß beide Phasen offensichtlich in enger Wechselwirkung zueinander stehen.
Untersuchungen in dieser Richtung werden in [4] beschrieben. Im Vordergrund steht die Verknüpfung von Navigationsaufgaben und Weltmodellierung. Die mobile Einheit wird als AMR (*autonomous mobile robot*) eingeführt, deren Hauptaufgabe das Durchmessen von Wegabschnitten ohne zusätzliche externe Hilfestellung ist. Zur Begriffsklärung werden hierzu folgende Definitionen gegeben [5], [6]:

> ***autonomous vehicles**: "simply defined, an autonomous vehicle must travel from one specified location to another with no external assistance" [5]*

> ***robots, mobile**: "a mobile robot is a free-roving collection of functions primarily focused on reaching a designed location in space" [6]*

Sehr wichtig ist in dieser Arbeit die Einordnung von AMR als Teil eines mehrschichtigen komplexen Regelkreises. Der Unterschied zu konventionellen Rechnersystemen wird in den wechselnden Umgebungsparametern gesehen. So arbeiten übliche Rechersysteme in

zumeist statischen Umgebungen, in denen die Problemlösung mittels vorgegebener Eingabedaten und Algorithmen erfolgt. Die Mobilitätsbedingung autonomer Systeme steigert die Varianz der Eingangsinformationen immens, zudem ist die autonome Einheit selbst verändernder Faktor innerhalb der Umgebung. Vereinfacht ausgedrückt, ist der AMR infolge seiner Bewegung in der Umwelt ein "Störfaktor", der zusätzlich und zum Teil praktisch nicht vorhersehbar die Vielzahl möglicher Regelkreise beeinflußt.
Im Bild 2.5 ist dieser Zusammenhang grafisch dargestellt. Ein Gedankenexperiment veranschaulicht die Grafik. Zum Zeitpunkt $t_1$ wird der interne Regler *R[..]* des AMR mit einer Anzahl Parameter *Par()* beaufschlagt. Mit großer Sicherheit ist die lokale Position des AMR ein Teil des Parametersatzes. Eine existierende Lösung der Regelaufgabe ist die Bewegung des AMR zum nächsten Wegabschnitt. Zum Zeitpunkt $t_2$ sei dieser erreicht.
Der Parametersatz für den Regler des AMR ist jetzt zumindest um die Ortsinformationen variiert worden

$$R_{t_2}[\{x+\triangle, y.., z..\} \in Par(t+\triangle)] \leftarrow R_{t_1}[\{x, y, z\} \in Par(t)]\ . \qquad (2.1)$$

Sehr wichtig scheint die Frage nach der Grenze der inkrementellen Schrittweite bei der Berechnung des Wegabschnittes zu sein. Inwiefern bestimmt diese Schrittweite die erforderliche Rechenleistung bzw. legt der grundsätzlichen Berechenbarkeit anhand eines Weltmodells Grenzen auf?

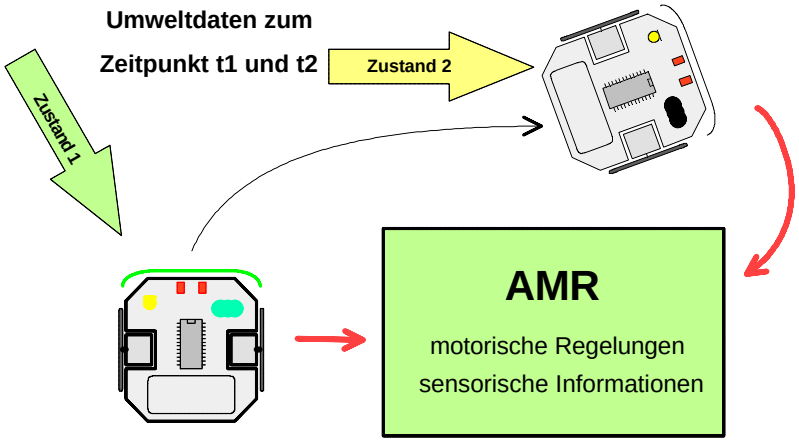



Bild 2.5 Komplexe zeitvariante Regelkreise eines AMR

In [4] wird dazu versucht, die bisher als grundlegend besprochenen Phasen 1 und 2 (Wegplanung und Fahrt) als eng miteinander verzahnte Prozesse in einem Ansatz zu behandeln. Wesentlicher Gegenstand dieser Arbeit ist die Implementation der

gewonnenen Ergebnisse und der Test unter praxisrelevanten Bedingungen. Der Ausgangspunkt dazu ist der Test in einer Simulationsumgebung und die abschließende Portierung auf dem realen System (Mobot-III).
Ohne auf die Einzelheiten der Umsetzung des in [4] beschriebenen Systems einzugehen, ist ein Scherpunkt der Realisierung die vorherige Simulation innerhalb eines Modells und die nachfolgende Umsetzung der Simulationsdaten auf die Hardware.
Ein solches Verfahren wird in einer Reihe von Variationen bei den unterschiedlichsten Realisierungsansätzen autonomer mobiler Systeme verwendet.

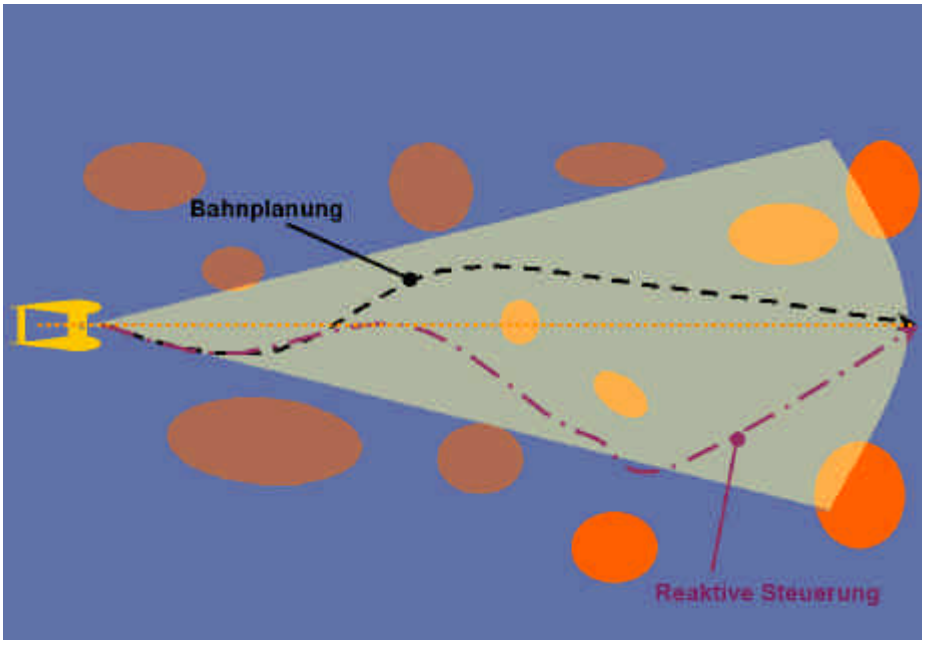



Bild 2.6 Bahnplanung beim Projekt "DeepC" der TU Ilmenau [55]

Einige Vorteile sind offenkundig. Die Simulation erfordert die mehr oder minder komplette Erfassung des Navigationsmodells in einer Berechnungsvorschrift (Programm). Bevor es zur Umsetzung der Verfahren auf eine Hardware kommt, mit allen Unwägbarkeiten und Störeinflüssen, die daraus resultieren, kann vorab die prinzipielle Funktionsweise sichergestellt werden. Die technische Umsetzung ist dann "nur" auf die Minimierung der unvermeidlichen Störquellen zu richten. Der Einwand, daß die potentiell unmögliche Störquellenminimierung eines unglücklich gewählten Mechaniksystems ein solches Projekt zum Scheitern verurteilt, soll lediglich die Probleme eines komplexen Systems aus Software, Elektronik und Mechanik besser verdeutlichen.
In [7] ist ein Versuch zur Erfassung von potentiellen Störgrößen und deren späterer Minimierung unternommen worden. Hier wird unter der ausschließlichen Nutzung bekannter und dadurch hinreichend stabiler Verfahren ein Simulationssystem

beschrieben, so daß mit Hilfe grafischer Ausgaben die Steuerung eines Unterwasserfahrzeuges visuell sichtbar gemacht wird.
Ein Schwerpunkt dieser Arbeit lag auf der Nutzung von Standardrechentechnik (PC) und der Einsatz verbreiteter Software, Open-GL als Visualisierungstool um möglichst effizient und kostengünstig Ergebnisse zu erzielen.
Eine weitere Aufgabe besteht darin, die Kopplung von internen Modelle, Fahrzeugmodelle, Störmodelle (alle in Form mehr oder minder komplexer Regelkreise) mit den Fähigkeiten menschlicher Operatoren zu verknüpfen.

Bild 2.7 "Honda-Mann" als Beispiele einer komplexen Operator/Rechner-Kombination [40]

Der Sinn einer solchen Verknüpfung wird in der Abstraktion von typischen Steuerverhalten menschlicher Operatoren und deren Kategorisierung nach spezifischen Eigenschaften gesehen. Offenbar liegt hier die Vermutung des Vorhandenseins intuitiver Steuermechanismen, die sich so ohne weiteres nicht aus mathematischer Modellierung ergeben, zugrunde. Begrenzt wird ein solches Vorhaben lediglich durch die Qualität der eingesetzten Modelle. Insbesondere müssen für möglichst realitätsnahe Simulation- en Störgrößen in die Modellierung einfließen. Da solche Störungen nur höchst unvollkommen der Realität nachempfunden werden können, sind die Grenzen der Simulation absehbar.

Dennoch liefern komplexe Simulationen brauchbare Ergebnisse, die in Folge weiterer Untersuchungen nützlich sind.
Das Gesamtsystem "Simulation und Operator" ist in diesem Zusammenhang als Weltmodell zur Führung und Lenkung autonomer Fahrzeuge zu interpretieren. Der Operator übernimmt hier die Funktion der *globalen Intelligenz* die in der Zusammenfassung aller Daten spezifische Steuerbefehle erteilt die dann als Aufschaltgröße in die eigentliche Fahrzeugmodellierung einfließen.
Als Beispiel eines solchen "Weltmodell-basierten" Systems ist der Roboters Honda P3 (siehe Bild 2.7) anzusehen. In diesem Falle werden extreme Forderungen sowohl an die mechanische Konstruktion als auch an die notwendige Steuertechnik gestellt.
Der Roboter ist als zweibeinige gehfähige Maschine (Biped) mit teilautonomer Steuertechnik ausgeführt. Mehrere Gyroscope, Beschleunigungssensoren und Kraftaufnehmer dienen der Lagedetektion und Steuerung. Mittels Videokameras wird eine Umwelterkennung realisiert, ein WLAN Kommunkationssystem fordert, falls nötig, zusätzliche Rechenleistung an (bzw. Operatoreingriffe).
Die Web-Site *(www.world.honda.com/robot*) informiert über weitere Einzelheiten des Systems. Eindrucksvolle Filmsequenzen belegen die Leistungsfähigkeit des Roboters. Erklärtes Ziel dieser Entwicklung ist ein *Humanoid Robot*, der zusammen mit menschlichen Partnern interagiert. Die Konstruktion als Humanoid soll wohl die unterschwelligen Vorbehalte gegen "nichtmenschliche" Kunstwesen minimieren. Betrachtet man einige der Filmsequenzen, die den Roboter in Aktion zeigen, so scheint dieses Vorhaben gelungen. Die Bewegungen wirken flüssiger und weniger abgehackt als die schnellen, hochoptimierten Positionierbewegungen üblicher Industrieroboter.
Der enorme Aufwand, der für diesen Roboter notwendig ist, offenbart einige zukünftige Möglichkeiten humanoider Systeme als Ergänzung menschlicher Fähigkeiten. Andererseits ist der Aufwand ein mobiles System mit zwei Beinen auszurüsten und zusammen mit menschlichen Kooperationspartnern einzusetzen, geradezu absurd hoch, so daß in absehbarer Zeit kein sinnvoller Rationalisierungseffekt zu erwarten ist. Auch der Drang nach humanoiden Konstruktionsformen ist nur teilweise nachvollziehbar. Normale Arbeitsgeräte werden auch nicht zwangsläufig menschlichen Körperformen nachempfunden; ein einfacher Hammer besteht aus einem Stiel und einem Metallkopf und ist nicht in Form einer Hand mit Faustkeil ausgeführt.
Es bleibt jedoch festzustellen, daß die dauernde Integration von Robotern in eine "menschliche" Umgebung, d.h. die Interaktion von Robotern mit Personen, ohne ein dem menschlichen Empfinden angepaßtes Mensch-Maschinen-Interface schwierig erscheint.

Derzeitige Kommunikationsverbindungen sind noch zu speziell und in ihren Reaktionen wenig flexibel, um ohne tiefere Kenntnis im Alltag gebrauchsfähig zu sein.

### 2.2.2 Verhaltensbasierte Steuerung

Das Dilemma der ungelösten "Weltmodellierung" führte schon recht früh zu alternativen Lösungsansätzen. Vergleichsweise einfache Reflexe sollten mechanische Konstruktionen zu neuen "intelligenten" Reaktionen befähigen.
Mit einer gewissen Berechtigung ist die derzeitige Diskussion von welt- bzw. verhaltensbasierten Lösungsansätzen erst in den letzten Jahren sinnvoll geworden. In den 50-er bzw. 60-er Jahren stand eine derartige Diskussion hauptsächlich wegen der begrenzten Leistungen elektronischer Computer noch nicht zur Debatte.
Wie kann nun ohne leistungsfähige Rechentechnik eine Maschine "intelligent" gemacht werden? "Kybernetik" ist eines der notwendigen Zauberworte dazu. Was verbirgt sich hinter diesem Ansatz?
Die Versuche von Ivan Petrowitsch Pawlow zur Konditionierung eines Hundes sind heutzutage beinahe als Allgemeinwissen einzustufen. Das Interessante daran ist die Herausarbeitung charakteristischer Verhaltensmuster und die Bezugnahme auf entsprechende Umweltreize. Der gewählte Versuchsaufbau mag heute archaisch anmuten, zur Gewinnung und Verifizierung der Ergebnisse war er ausreichend. In [10] ist eine detaillierte Versuchsbeschreibung zu finden.
Stellt sich natürlich die Frage, was ein solcher Versuch mit der verhaltensbasierten Steuerung von Robotern zu tun hat? Im Grunde geht es um die einfache Verknüpfung von Reiz und Reaktion, die Pawlow mit seinem Hund demonstriert hat.
Umweltreize werden (vorprogrammierten) Verhaltensweisen zugeordnet. Unter einem solchen Aspekt betrachtet, reduziert sich die komplizierte Weltbild-Modellierung auf die Verknüpfung einfacher Verhaltensweisen. Die provokante Frage: "Wenn das so einfach ist, warum sind wir dann noch nicht weiter?", darf zwar gestellt werden, erst später soll darauf eingegangen werden.
Diese abstrakte Idee ist in ihrer Einfachheit und Eleganz verblüffend. Die ersten praktischen Versuche dazu hat Walter W. Grey unternommen, die auf Untersuchungen von Norbert Wiener basierten [11], [12].
In der praktischen Realsierung dieser Überlegungen entstand ein dreirädriger Roboter der nur mit einer sehr einfachen Steuerung ausgerüstet, vermeintlich intelligente Verhaltensweisen zeigte. Das Fahrzeug ist in der Lage, auf Umweltreize, z.B. Lichtquellen, zu reagieren. Werden mehrere dieser Fahrzeuge mit Lichtquellen ausgerüstet, sind

Szenarien beobachtbar, die zwar teilweise chaotisch wirken, dennoch aber regelbasiert sind.

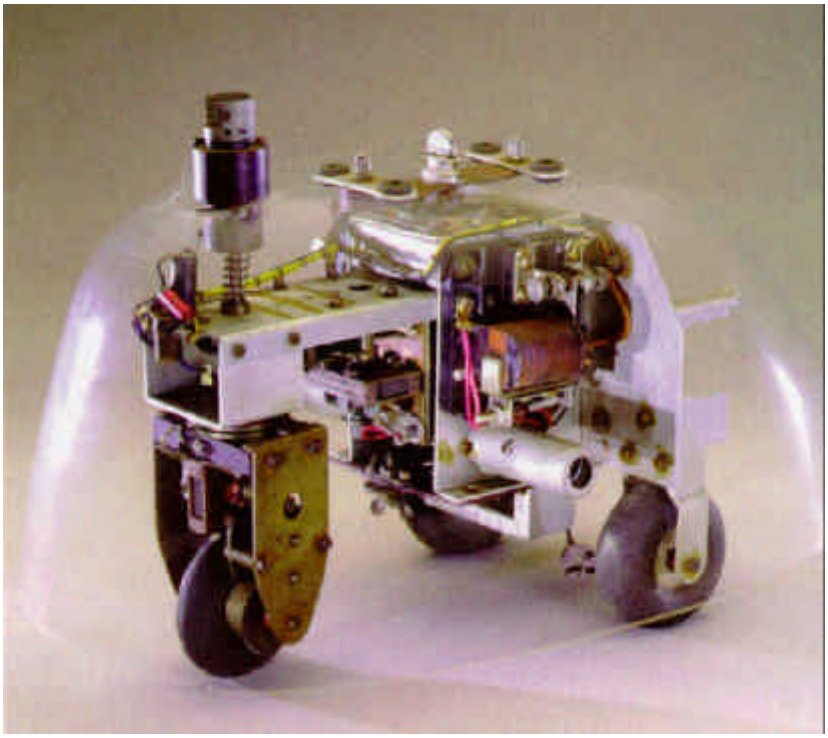


Bild 2.8 Kybernetische Schildkröte "Elsie" [11]

Grundlage dieser ersten Versuche waren Zielsuch- bzw. Fluchtstrategien, wie sie zum Teil auch im Tierreich verbreitet sind.
Das elektronische Gefährt versuchte mit Hilfe seines Sensors, Lichtquellen auszumachen und diese daraufhin anzusteuern. Mittels vorgegebener Schwellwerte führte eine zu große Lichtintensität, z.B. infolge zu starker Annäherung an die Quelle, zu einer Ausweichreaktion des Fahrzeuges. Sind mehrere dieser Fahrzeuge mit Lichtquellen ausgerüstet, ergeben sich die bereits erwähnten "chaotischen" Fahrmuster.
Als ein Stück Technikgeschichte sei auch die Innenschaltung des "Steuercomputers" von Elsie dargestellt (Bild 2.9). Verblüffend ist aus heutiger Sicht die vergleichsweise "primitiv" anmutende Steuerschaltung mit lediglich zwei Elektronenröhren als aktive Komponenten.
Das wesentliche Merkmal dieser und anderer ähnlicher Experimentalroboter ist die unmittelbare Verknüpfung von Umweltreiz und zugeordnetem Verhalten.
Spätere Versionen dieser kybernetischen Fahrzeuge verfügten über eine große Anzahl unterschiedlicher Sensoren und eine breite Palette differierender Verhalten. Entsprechend variabel wurden auch die beobachtbaren Reaktionen. Zusammenfassend läßt sich aus diesen Experimenten eine grafische Struktur verhaltensbasierter Systeme ableiten.

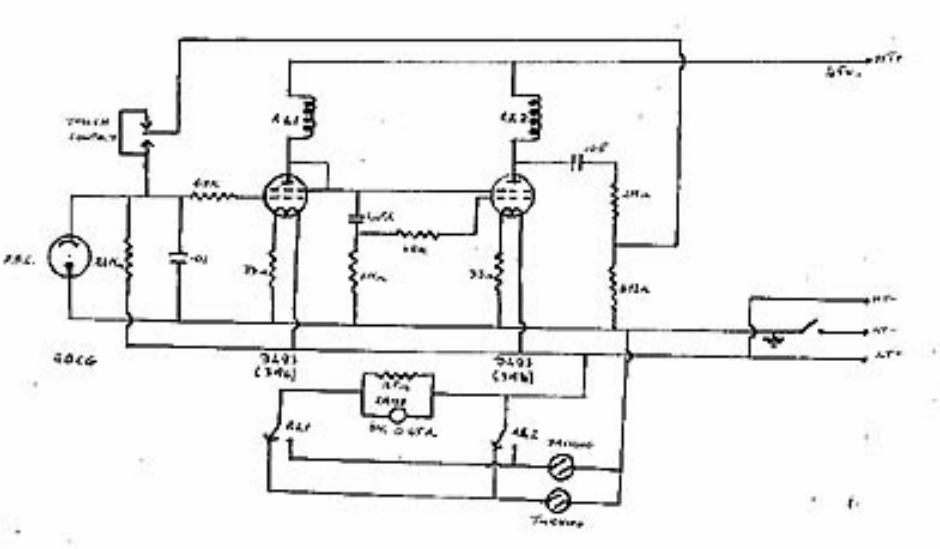


Bild 2.9 Steuerelektronik von "Elsie" [11]

Wie in der Abbildung 2.10 zu sehen, werden spezifische Umweltreize an bestimmte Sensoren geknüpft, z.B. an optische, akustische und taktile Sensoren. Jeder dieser Sensoren steuert ein damit verbundenes Verhalten.

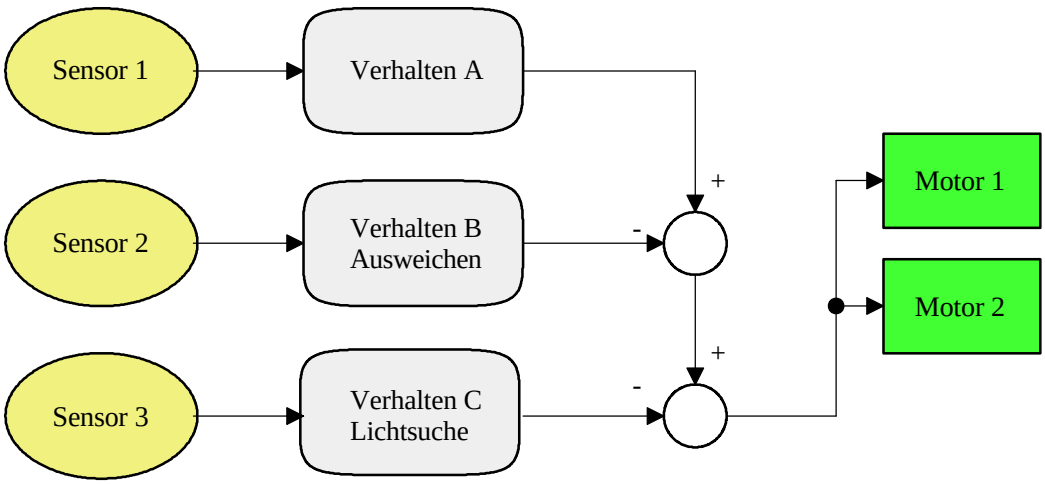



Bild 2.10 Subsumptionsarchitektur verhaltensbasierter Roboter

Damit die Maschine beim gleichzeitigen Auftreten mehrerer (aller) Umweltreize nicht "verrückt" spielt, sind die Reize hierarchisch miteinander verknüpft. Diese als *Subsumption* [1] bezeichnete Eigenschaft stellt damit auch eine Priorisierung der verschiedenen Reize untereinander dar.

Ein Beispiel verdeutlicht das Verfahren. Die primäre Aufgabe eines verhaltensbasierten Roboters soll die Lichtsuche sein, d.h. der Roboter steuert einen Kurs, der ihn in Sichtweite der Lichtquelle bringen soll. Am einfachsten wäre eine kreisförmige Drehung auf der Stelle, bis ein Lichtsignal empfangen wird. Um Abschattungen zu umgehen, soll das Fahrzeug hin und wieder seinen Standort wechseln. Hat der Roboter das Lichtsignal erkannt, kann er schnurstracks darauf zu fahren. Es sei denn, ein für ihn unüberwindbares Hindernis, z.B. ein Glaswürfel, läßt ihn zwar Kontakt zur Lichtquelle halten, seine Bewegung aber nicht fortsetzen.
Ein zweiter Sensor detektiert das Hindernis und das dazugehörige Verhalten "übersteuert" die ursprüngliche Lichtsuche. Die Sinnhaftigkeit der Subsumptionsmethode wird augenscheinlich. Würde das ursprüngliche Verhalten nicht geändert, müßte der Roboter vor dem Glaswürfel verharren bis seine Batterien erschöpft wären. Mit Hilfe solcher an Sensoren gekoppelter Verhalten sind auch komplexere Reaktionsmuster darstellbar, siehe [19] Verhalten "Räuber und Gendarm" oder [41].
Je komplexer die gewünschten Verhaltensmuster werden, um so verwickelter wird die Aufstellung einer passenden Subsumptionsmethode. Das Modell gerät recht schnell an seine Grenzen. Damit scheinen die Skeptiker mit ihrer provokanten Fragestellung eingangs des Kapitels recht zu bekommen.

## 2.3 Komplexe Sensordateninterpretation

Kennzeichnend für die eben erläuterten Entscheidungsmodelle ist der hohe Grad der Abstraktion von Sensorsignalen. Im einfachsten Fall besteht im Beispiel ein Sensorsignal aus dem Vorhandensein, bzw. der Abwesenheit einer Lichtquelle. Zur Untersuchung formaler Eigenschaften dieser Systeme ist eine solche Abstraktion sinnvoll, in der Übertragung auf reale Applikationen, stößt dieses Vorgehen auf Schwierigkeiten. So sind reale Objekte selten mit Lampen, Leuchten oder anderen Signalmitteln ausreichend stark von der Umgebung abgegrenzt.
Unabhänging vom gewählten Sensor sind teilweise aufwendige Algorithmen notwendig, um aus dem Signal mit hinreichender Genauigkeit die erforderlichen Abstraktionen abzuleiten. In diesem Zusammenhang spielt die Spezialisierung des Detektionsapperates eine wichtige Rolle. Aus der Sicht des Anwenders stellt sich diese Auswahl als eine Gratwanderung zwischen Erkennungssicherheit und Spezialisierung der Sensortechnik dar.
Ein einfaches Beispiel zeigt das Dilemma. Transportroboter in Fabrikgebäuden bzw. Lagerhallen sind als relativ einfache Systeme konzipiert und auf Grund der zu

erwartenden Rationalisierungseffekte Objekt vieler Untersuchungen. Im wesentlichen besteht ihre Aufgabe darin, Materialien, Werkzeuge, Teile, Produkte, etc. zu transportieren. Damit sich der zusätzliche (menschliche) Steueraufwand gering hält, werden effiziente und zuverlässige Leitsysteme benötigt. Sehr sicher sind mechanische Systeme wie z.B. gleisgeführte Systeme. Ob der Transportroboter dabei direkt auf Schienen fährt, bzw. über einen Führungsmechanismus an einer mechanischen Leiteinrichtung entlang fährt, ist weniger wichtig. Kennzeichnend ist die starke Kopplung an den Verlauf der Transportwege und die eingeschränkten Möglichkeiten der Wegänderungen. Die mechanischen Leiteinrichtungen müssen oft bereits bei der Planung der Fabrik- bzw. Lagerhallen berücksichtigt werden und eine spätere Änderung ist mit hohen Kosten verbunden. Die Transportsicherheit ist jedoch sehr hoch und der Gefährdung der Umgebung kann durch (ortsunveränderliche) Schutzeinrichtungen als gering eingestuft werden.

Bild 2.11 Transportroboter James [28]

Flexibler sind Transportroboter, die sich mit Hilfe einfacher Markierungen, z.B. farbiger Leitlinien auf dem Boden der Lagerhallen bewegen können. Die mechanische Radlenkung durch Schiene oder Führungsstift wird durch einen elektronischen Sensor und dazugehörige Aktoren (Lenkung) ersetzt.
Je nach gewählter Markierungsart, optisch bzw. elektromagnetisch, sind Streckenänderungen deutlich einfacher als bei rein mechanischen Systemen möglich. Diese

Vereinfachung wird jedoch mit einem erhöhten Aufwand im Sensor erkauft. Gleichzeitig verstärkt sich der potentielle Gefährdungsgrad des Systems. Ohne mechanische Führung müssen weitere Sensoren zur Kollisionsvermeidung bzw. Gefahrenerkennung integriert werden. Im Ergebnis entsteht eine komplexe Detektorbaugruppe aus diversen spezialisierten Sensoren, z.B. optische Spurerkennung mit Ultraschallabstandssensor und/oder Laserscanner zur Hinderniserkennung. Mechanische Berührungsschalter mit "Notaus-Funktion" stellen sicher, daß schlecht erkennbare Hindernisse oder unvorsichtige Personen keinesfalls zu Schaden kommen.
Entsprechend der geforderten Sicherheit, ist der Aufwand an Spezialsensorik beinahe beliebig steigerbar. Derartige Speziallösungen widersprechen jedoch zum einen dem Grundgedanken universeller Roboter, und zum anderen bedeuten sie erheblichen Aufwand mit der Fixierung auf nur eingeschränkte Erkennungen.
Unter dem Aspekt immer wirklichkeitsnäherer Anwendungen, z.B. fahrerlose Transportsysteme im Straßenverkehr (und hier sind keine besonderen Leiteinrichtungen realisierbar), muß die Sensorik an die Aufgabenstellung adaptierbar werden.

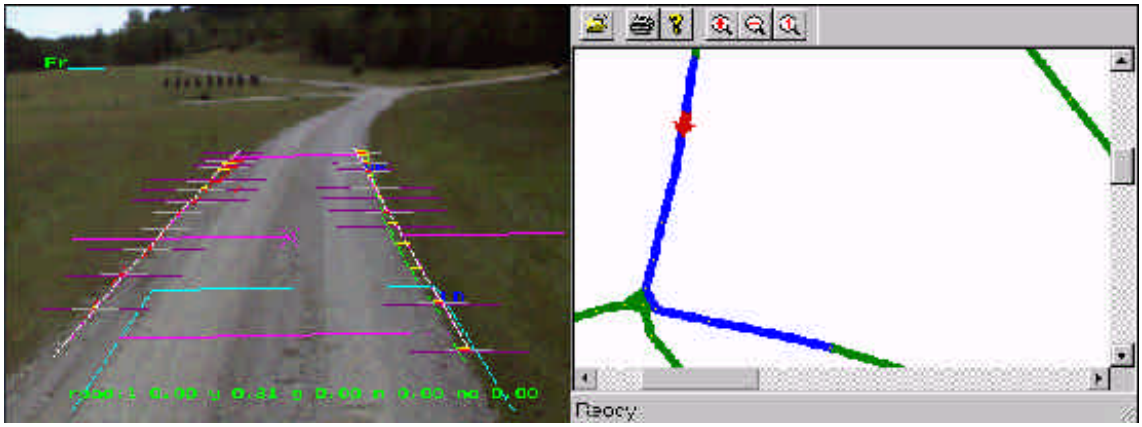


Bild 2.12 Maschinelles Sehen beim Roboter [29]

Um vorerst beim Transportroboter zu bleiben, bedeutet dies, daß die Entwicklungsrichtung weg vom Spezialsensor hin zum programmierbaren Detektor führt. In [29] werden (hoch)komplexe maschinelle Sehverfahren beschrieben, die anhand von Kamerabildern in Echtzeit zuverlässig Fahrspuren erkennen, Hindernisse auf dem Fahrweg detektieren und teilautonom Ausweichstrecken ermitteln.
Die Abbildung 2.12 zeigt ein solches realisiertes System mit mehreren Kameras zur Fern- bzw. Weitwinkel-Bildaufnahme. Die erforderliche Rechenleistung zur Echtzeitbildverarbeitung wird durch ein leistungsfähiges Multiprozessorsystem bereitgestellt.
Diese Beispiele erhellen schlaglichtartig die Problematik und Komplexität komplexer Sensordateninterpretation. Für die Anwendung im Low-Cost-Robotersystem MauSI

kommen Mehrkamerasysteme und Multidatenprozessoren nicht in Betracht, die ausschließliche Verwendung konventioneller Sensorik grenzt den Einsatzbereich jedoch unzulässig ein. Ein programmierbarer Kamerasensor und einfache Bildauswertealgorithmen stellen für MauSI im Vergleich zu anderen Roboterplattformen einen erheblichen Leistungszuwachs in diesem Robotersegmet dar.

## 2.4 Lösungsansätze für ein reales System

In allen diesen Untersuchungen wird dem Steuermodell die Aufgabe eines internen Beobachters zugewiesen, bzw. diese Aufgabe einem externen Operator zugewiesen. Aus dem im Ausdruck (2.1) angegebenen Zusammenhang wird die Notwendigkeit einer Fehlerschranke, die dem Operator (bzw. dem Weltmodell) Abschätzungen zum aktuellen Zustand (Verhalten) der mobilen Einheit ermöglicht. In [8] wird dieser Zusammenhang in Form einer Funktion zur Fehlerminimierung angegeben

$$y_{t+k} = f(\varepsilon_t) = f(x_t - \tau_t) \; . \tag{2.2}$$

Der Fehler $\varepsilon = x_t - \tau_t$ ist das Eingangssignal für die Steuerung, mit dem Ziel, diesen Fehler zu minimieren. Dabei ist $x_t$ der aktuelle Systemstatus zum Zeitpunkt $t$ und $\tau_t$der verlangte Zustand. Für den Operator ergibt sich die notwendige Aktion $y_{t+k}$ mit dem Zeitschritt $t+k$. Das erforderliche Steuermodell verbirgt sich hinter der Funktion $f()$ und ist zunächst unbekannt.

Bei den bislang erläuterten Systemen ergeben sich offenbar Probleme, diese Funktion zur Steuerung der mobilen Einheiten in einem Modellraum zu erfassen. Sehr schnell steigt die Komplexität der erforderlichen Rechenleistung zur kontrollierten Führung der mobilen Systeme. Es kommt zu einem Widerspruch zwischen erreichbarer Autonomie der Einheiten und materiellen Aufwand für die Einzelgeräte.

Um diesem Dilemma zu entgehen wird der Versuch unternommen, für die mobile Einheit eine Minimalkonfiguration zu bestimmen, für die genügend große Werte für den Zeitschritt gefunden werden können.

Ausgehend von allgemeinen Eigenschaften ergeben sich folgende Realisierungsschwerpunkte:

- autonome Bewegung im Raum, hier unter der Vereinfachung nur $R^2$ (zweidimensionaler Raum) innerhalb eines zu bestimmenden Weg-Zeit-Fensters - **Teilautonomie des Roboters**

- Überwachung der Fehlerschranken durch einen lokalen Operator in der mobilen Einheit
- Kopplung der Systeme an einen übergeordneten Operator zur globalen Pfadplanung
- Bereitstellung der Möglichkeit, mehrere Systeme zu führen
- Begrenzung des Rechenaufwandes der lokalen Einheit durch Zerlegen des Gesamtweges in Teilabschnitte (diskrete Teilwege)
- Bereitstellung ergänzender Informationen (Prognosewerte) zu den diskreten Teilwegen .

Zur Lösung dieser formalen Vorgaben wird ein mechanisches System aufgebaut, eine elektronische Steuereinheit konzipiert und die grundlegenden softwaretechnischen Voraussetzungen geschaffen, um ein mobiles autonomes System zu erhalten, dessen wesentliche Parameter (Rechenleistung, Softwareschnittstellen) ausreichen, eine teilautonome lokale Intelligenz zu steuern und zu programmieren.
Die dazu notwendigen Vorausetzung unterteilen sich in folgende Bereiche:

- Mechanikplattform zur Bewegung im Raum ($R^2$), *differential drive* Antriebskonzept
- Elektronikbaustein (Mikrocontroller) zur Ansteuerung der Sensoren und Aktoren, inklusive Stromversorgung
- Softwarekonzept mit gekapselten Softwaremodulen für Sensoren und Aktoren
- Minimalaustattung mit Distanzsensoren, Beschleunigungsmessern (zweiachsig), Funkmodulen sowie optional Kamera, Tastern und Solarzellen
- extern rekonfigurierbare Software auf dem Steuercontroller, d.h. während des Betriebes müssen neue Funktionalitäten, Algorithmen etc. nachinstalliert werden können, z.B. per Funk
- Konzeption der Elektronikeinheit (und teilweise auch der Mechanik) als *low-cost*-System .

# 3 Mobile Einheit mit variablen Eigenschaften

## 3.1 Konzeption

Im Ergebnis der bisherigen Untersuchungen und Überlegungen ergibt sich ein enger Zusammenhang zwischen Mechanik, Elektronik und Software. Die sehr unterschiedlichen Antriebsprinzipien, z.B. Fahren, Schreiten oder Fliegen, erfordern zum Teil sehr hohe Aufwendungen in der Elektronik und der Software, um die notwendigen Grundfunktionen zu realisieren. Als Extrembeispiel wäre hier die Lageregelung eines fliegenden Roboters zu nennen. In einem solchen Fall sind erhebliche Ressourcen in der Computerhardware für die komplexen Lageberechnungen vorzuhalten, die der eigentlichen "Intelligenz" des Roboters nicht mehr zur Verfügung stehen. Zu berücksichtigen sind auch die erforderlichen Steueraktuatoren, z.B. Rudermaschinen etc., die als Kostenfaktoren bzw. Energieverbraucher das System verteuern.
Etwas weniger kritisch stellt sich die Problematik bei erdgebundenen Fahr- bzw. Schreitsystemen dar. Speziell bei den schreitenden Robotern sind in den letzten Jahren große Fortschritte erzielt worden [14].
Ein weiterer wichtiger Aspekt für die Auswahl des Antriebskonzeptes ist der minimale Energieverbrauch, der zur Aufrechterhaltung des stationären Zustandes des Systems benötigt wird. Stationär ist der Roboter genau dann, wenn er keine Kommandos zur Ortsveränderungen aus seinen internen Entscheidungsabläufen bzw. vom externen Operator erhält. Im Beispiel eines radgetriebenen Fahrzeuges wäre der Energieaufwand beim Stillstand der Antriebe beinahe null (nur der Ruhestrom der Ansteuerelektronik ist zu berücksichtigen).
Der Energieverbrauch fliegender bzw. schreitender Roboter ist hier ein die Missionsdauer stark beeinflussender Faktor.
Solange die Robotersysteme nur Demonstrationscharakter tragen, spielt die Robustheit der Konstruktion nur eine untergeordnete Rolle. Innerhalb der vorgegebenen Modellwelt müssen reale Störungen, wie z.B. Witterungseinflüsse oder widrige Bodenbeschaffenheiten (Schlamm, Sand, etc.) kaum beachtet werden. Soll der Roboter jedoch auch in der realen Welt arbeiten, muß die Fahrwerksauslegung an die zu erwartenden Bedingungen angepaßt werden. Problematisch sind hier Roboterkonstruktionen, welche auf dem Prinzip des Differentialantriebs (*differential drive)* beruhen. In Modellwelten wird die Instabilität des zweirädrigen Antriebes meist durch ein Stützrad kompensiert. Eine

einfache mechanische Vergrößerung der Abmaße des Differentialantriebs des Roboters scheint hinsichtlich der praktischen Eignung im Alltag zweifelhaft.
Trotz der kurz umrissenen Probleme im Roboterdesign werden die ersten Versuche mit einem Differentialantriebssystem unternommen. Alle anderen Verfahren wie Schreiten oder Fliegen werden aus technischen (zu kompliziert) bzw. energetischen (unklare Energieversorgung) Gründen verworfen.

## 3.2 Mechanikplattform

### 3.2.1 Miniaturroboter mit Differentialantrieb

Die Vorteile des Differentiantriebes sind für einfache Robotikanwendung so augenscheinlich, daß die vermeintlichen Nachteile vorerst in Kauf genommen werden.
Einer der wesentlichsten Vorzüge besteht in der ausgezeichneten Manövrierbarkeit der Mechanik und der relativ geringen "Steuerintelligenz", die dazu erforderlich ist.

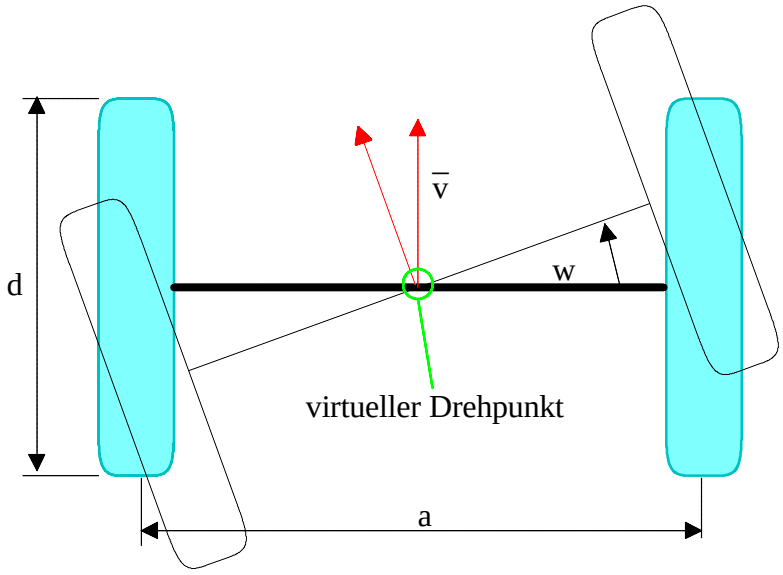



Bild 3.1 Antriebsprinzip des Differentialantriebes

Kernpunkt des Antriebes sind zwei unabhängig voneinander ansteuerbare Motoren. In der Praxis haben sich Gleichstrommotoren (*DC-motor*) oder Schrittantriebe (*stepper motor*) bewährt.
Schrittantriebe bieten eine bessere Präzision bei der Programmierung unterschiedlicher Drehgeschwindigkeiten und können im Rahmen ihrer Schrittauflösung genau gesteuert werden. Der Schrittmotor ist in dieser Anwendung sowohl Antriebselement als auch

Sensor. Ihm können exakte Wegstrecken vorgegeben werden. Positionierfehler sind nur noch durch den Schlupf der Räder auf dem Boden abhängig.
Der wesentliche Nachteil des Schrittmotors liegt in seinem schlechten Wirkungsgrad bedingt. Übliche Bipolar-Motoren erzeugen ihr Drehmoment durch umlaufende Magnetfelder, die den im Inneren des Motor befindlichen Anker auslenken. Der Anker ist gepolt (starker Dauermagnet). Die Bewegung von Ankermagnetfeld und äußerem Magnetfeld erfolgt immer synchron. Es dürfen nur sehr geringe Winkeldifferenzen zwischen Ankerfeld und äußerem Magnetfeld auftreten, ansonsten gerät der Motor außer Tritt. Überholt das äußere programmierte Feld den Anker um mehr als einen Winkelschritt, kann der Schrittmotor sogar kurzzeitig rückwärts laufen. In der Praxis bedeutet das für den Antrieb, daß er nur kleine dynamische Drehmomentänderungen verkraften kann. Die Dimensionierung muß so sicher sein, daß das maximal nötige Drehmoment unter allen Betriebsbedingungen erreicht wird. Besonders nachteilig bei diesem Verhalten ist die Tatsache, daß Schrittfehler ohne zusätzlichen Meßaufwand, z.B. optische Encoder o.ä., von der Steuerelektronik nicht erkannt werden können.

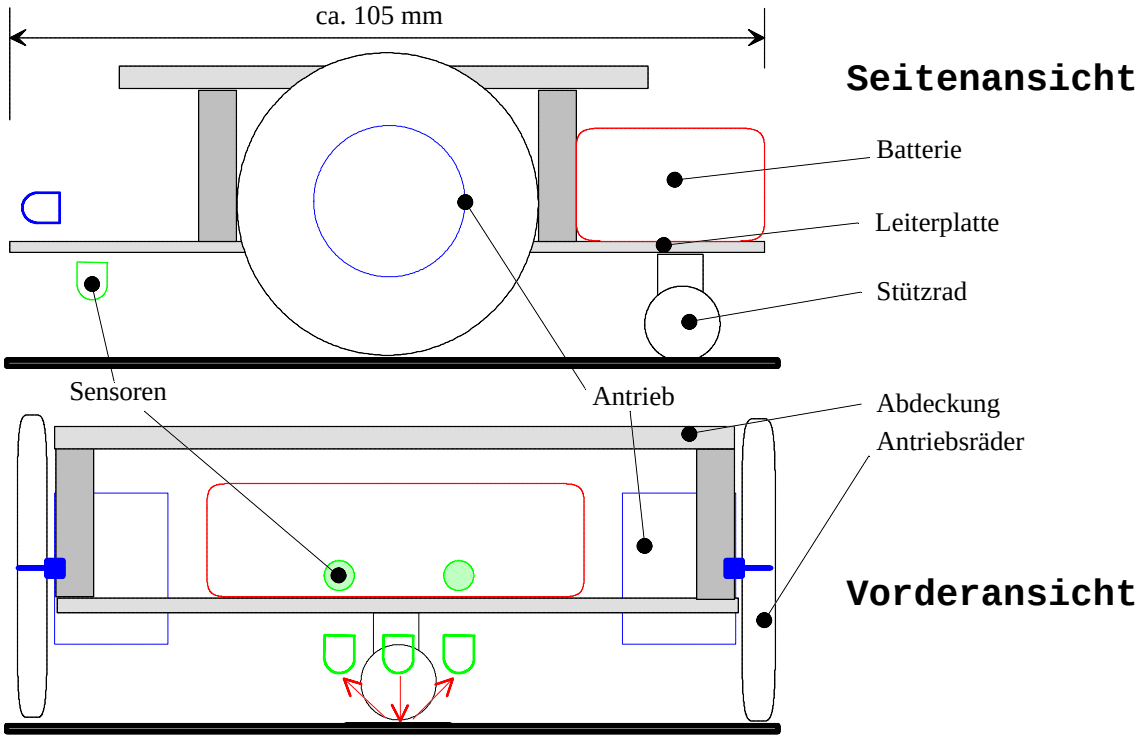



Bild 3.2 Aufbauprinzip des Miniaturroboters

Der schlechte Wirkungsgrad des Motors resultiert aus Ansteuerverfahren zur Erzeugung des umlaufenden Magnetfeldes. In die zwei Wicklungen des Bipolarmotors werden mit

Hilfe von Treiberschaltkreisen die Phasenströme eingeprägt. Die Treiberschaltung regelt die Ströme in den Wicklungen über Stromchopperung. Der Spannungsabfall an Meßwiderständen wird als Maß für den eingeprägten Strom gewertet. Infolge der Verluste im Magnetkreis von Wicklung, Statorpaket und Anker wird ein großer Anteil der eingeprägten Leistung in Wärme umgesetzt.
In bestimmten Anwendungen kann der Schrittmotor in Ruhe nicht völlig stromlos geschalten werden, da entweder das Haltemoment zu klein ist, bzw. bei Mikroschrittansteuerung wegen der Nichtlinearität von Spulenstrom und Drehmoment ein Reduzieren der Phasenströme. selbst bei gleichem Stromverhältnis, Schrittfehler nach sich zieht. Damit fällt die Entscheidung zum Antrieb des Roboters zugunsten von Gleichstrommotoren.
Die Skizze 3.2 zeigt die wesentlichen Komponenten und den prinzipiellen Aufbau des Miniaturroboters. Wegen der geringen Abmaße und der relativ kleinen Masse des Roboters kann die Leiterplatte des Elektronikblocks in das mechanische Konzept aufgenommen werden. Die Leiterplatte und die Abdeckung werden mit Distanzbolzen verschraubt. Die beiden Motorträger sind an den Distanzbolzen befestigt und stabilisieren den Roboter zusätzlich.

### 3.2.2 Dimensionierung des Antriebes

Aus den bereits genannten Gründen scheiden Schrittmotoren als Antriebsquelle für den Roboter aus. Bevor die diversen Motortypen auf ihre Brauchbarkeit untersucht werden, ist eine Abschätzung der erforderlichen Antriebsleistung notwendig. Da die Ermittlung der einzelnen Fahrwiderstände und Schätzung von Reibungsverlusten auf ebener Strecke mit zu vielen Fehlerquellen behaftet ist, wird zur Abschätzung der minimal notwendigen Antriebsleistung eine Testfahrt des Roboters auf einer Rampe zugrundegelegt. Die Rampe wird so dimensioniert, daß alle anderen Fahrwiderstände sehr viel kleiner sind als die Fahrleistung, die zur Auffahrt auf die Rampe benötigt wird.
Für die Berechnung der Drehmomente wird das maximale Gewicht des Roboters mit 200g festgelegt. Dieses Robotergewicht müssen die Antriebe eine Rampe mit 15 Grad Neigungswinkel hinaufbefördern. Damit gilt:

$$\vec{F}_r = \sin\alpha \cdot \vec{F}_g \approx 500 \text{ mN}. \qquad (3.1)$$

Beim Maximalgewicht von 200 g ist eine Vortriebskraft von ca. 500 mN nötig. Bei einem Durchmesser der Antriebsräder von ca. 42 mm, bedeutet dies ein minimales Drehmoment von 5,25 mNm pro Rad.

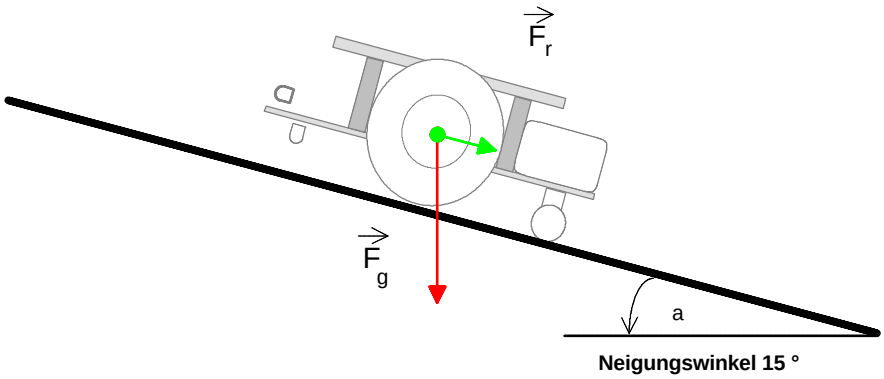



Bild 3.3 Abschätzung der Antriebsskräfte des Miniarturroboters

Aus der Vielzahl der möglichen Typen werden Miniaturmotoren FTB 20-08 der Fa. Feintechnik-Bertsch (*www.ftb-bertsch.de*) ausgewählt. Die Spezialmotoren sind infolge ihres besonderen Wicklungsschemas außerordentlich flach aufgebaut. Die Motoren sind mit einem eisenlosen Rotor ausgerüstet. Der Rotor ist als frei tragende Spule auf die Abtriebswelle aufgebracht. Im Gegensatz zu üblichen Motoren sind die Permanentmagnete zur Felderzeugung an der Stirnseite bzw. an der Bodenplatte angeordnet. Damit wird eine Bauhöhe, inklusive des direkt angeflanschten Getriebes, von lediglich 20 mm erreicht. Beeindruckend ist auch der Wirkungsgrad von ca. 80%.
Die ausgewählten Motoren haben ein empfohlenes Dauerdrehmoment von 0,4mNm. Bei der Getriebeuntersetzung 139:1 mit dem Wirkungsgrad von 66% ergibt sich ein maximal zulässiges Dauerdrehmoment von:

$$M_{out} = n \cdot M_{in} \cdot \eta = 37\,\text{mNm}. \qquad (3.2)$$

Dieser Wert reicht mit Sicherheit aus, um dem Roboter einen sicheren und zuverlässigen Antrieb zu ermöglichen. Gleichzeitig wird damit auch eine genügend große Reserve für die Motorregelung und den Betrieb des Roboters bei kleinen Geschwindigkeiten bereitgestellt. Die genauen Motordaten sind im Anhang aufgeführt.

## 3.3 Elektronikkonzept

### 3.3.1 Stromversorgung

Mit der Auswahl der Antriebsmotoren sind wesentliche Energieverbraucher definiert, so daß die Systemstromversorgung dimensioniert werden kann. Zuerst wird anhand der Stromaufnahme wichtiger Teilkomponenten eine überschlägige Energiebilanz ermittelt:

| | Sleep-Mode | Standby-Mode | Aktiv-Mode |
|---|---|---|---|
| Mikrocontroller | 2 mA | 4 mA | 32 mA |
| Antriebsmotoren | 0,010 mA | 0,100 mA | 45 mA |
| Funkmodul | 0,001 mA | 4 mA | 21 mA |
| Infrarotsensoren | 0,001 mA | 0,001 mA | 6,5 mA |
| Soundausgabe | 2 mA | 2 mA | 10 mA |
| Videokamera | 0,2 mA | 0,200 mA | 30 mA |
| Summe | ca. 4,2 mA | ca.10,3 mA | ca.145 mA |

Bild 3.4 Energiebilanz wichtiger Funktionsgruppen

Die Stromaufnahmen der einzelnen Module gelten für eine Betriebsspannung von 5 Volt. Bei einer maximalen Stromaufnahme von 150 mA ist ein Betrieb aus Batterien bzw. Akkumulatoren problemlos möglich.
Für die Erzeugung einer stabilen Versorgungsspannung sind verschiedene Verfahren bekannt.

**Variante 1:**
Die einfachste und preisgünstigste Variante besteht im Einsatz eines Linearreglers. Derartige Bausteine sind in großer Zahl verfügbar. Als so genannte *low-drop-voltage-regulators* (LDO) benötigen sie zur korrekten Funktion eine Eingangsspannung, die etwa 250...500 mV über der gewünschten Ausgangsspannung liegt, d.h. es werden min. 5,2 V Batteriespannung (Akkuspannung) bei 5 V Systemspannung benötigt.
Die Ladeschlußspannung von NC-Akkus beträgt etwa 1 V. Zur Versorgung eines Linearreglers wären also min. 6 NC-Zellen nötig. Eine NC-Zelle der Größe LR6 (Mignon) wiegt ca. 20 g. Der Akkupack zur Stromversorgung aus Standardzellen wöge dementsprechend 120 g, das entspräche 60 % der spezifizierten Gesamtmasse des Roboters. Ein Einsatz von kleineren Knopfzellen ist unumgänglich. Damit reduziert sich

die verfügbare Kapazität von 1100 mAh (LR6 Zelle) auf etwa 150 mAh für Knopfzellen bei gleichzeitiger Verringerung der Masse auf ca. 50 g.
Der Zusammenhang von Akku- bzw. Batteriekapazität mit der Gesamtmasse des anzutreibenden Roboters läßt sich als Masse-Kapazitäts-Index ($C_M$) ausdrücken und stellt damit ein gutes Maß für die Leistungsfähigkeit der Energieversorgung dar:

$$C_m = \frac{C_{akku}\,[mAh]}{(M_{Akku} + M_{Robot})\,[g]} \cdot \qquad (3.3)$$

Beim Einsatz von sechs LR6-Zellen beträgt $C_M$ = 4,1. Dabei wird allerdings die angenommene Gesamtmasse von 200 g deutlich überschritten. Werden Knopfzellen benutzt, sinkt der Wert für $C_M$ auf 0,75.
Ein weiterer Aspekt ist der Wirkungsgrad der Stromversorgung. Bei vollen Akkus, ca. 7,2 V Eingangsspannung, fallen etwa 2,2 V am Regler ab. Im Aktiv-Mode entstehen damit Verluste von ca. 30 %. Der Wirkungsgrad der Stromversorgung wäre dann minimal 70 % und maximal 83 %. Das ist recht ungünstig für ein autonomes System, bei dem überflüssige Verluste schädlich auf die Gesamtperformance wirken.

**Variante 2:**
Der Linearregler wird durch einen Schaltregler ersetzt. Geschaltete Gleichspannungswandler ersetzen die lineare Regelung der Ausgangsspannung durch ein getaktetes System von Schaltern (Schalttransistor, Freilaufdiode) und Energiespeichern (Induktivität, Kondensator) [15].
Denkbar sind *step-down* bzw. *step-up*-Konverter. Im ersten Fall wird die Versorgungsspannung aus der höheren Eingangsspannung gewonnen, im zweiten Fall kann eine niedrigere Eingangsspannung "heraufgesetzt" werden. Beide Verfahren arbeiten mit Wirkungsgraden um 90 %.
Die Entscheidung, welches Verfahren beim Roboter eingesetzt wird, bestimmt der Masse-Kapazitäts-Index. Das *step-down*-Verfahren nutzt die Knopfzellen mit 7,2 V Eingangsspannung als Energiequelle, der Masse-Kapazitäts-Index bleibt unverändert ($C_M$ = 0,75), der Wirkungsgrad erreicht jedoch ca. 90 %.
Wird jedoch ein *step-up*-Konverter eingesetzt, können zwei LR6 NC-Zellen genutzt werden, $C_M$ steigt somit auf 5,5. Werden für spezielle Zwecke Alkaline Zellen, z.B. Energizer No.91 mit einer Kapazität von 2800 mAh verwendet, steigt $C_M$ auf 14 (Lithum-Zellen mit noch höherer Kapazität ergeben $C_M$ = 16).
Diese deutlich bessere Leistungsbilanz hat jedoch ihren Preis. Anstelle des einfachen preiswerten Linearreglers in bewährter Schaltungstechnik erfordert ein hocheffizienter

*step-up*-Regler neue schaltungstechnische Lösungsansätze. Ein elektronischer Schalter steuert den Stromfluß durch die Induktivität. Die Spannungsspitze, die beim Sperren des Schalters an der Induktivität auftritt, lädt den Kondensator auf. Der Regler sorgt dafür, daß in Abhängigkeit der Belastung von $V_{cc}$ der Schaltet getaktet wird.

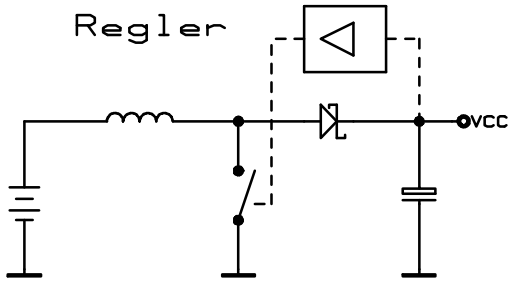



Bild 3.5 Funktionsprinzip eines step-up-Reglers

Die Prinzipschaltung ist in Abbildung 3.5 zu sehen. Die exakte schaltungstechnische Realsierung zeigt Bild 3.6.

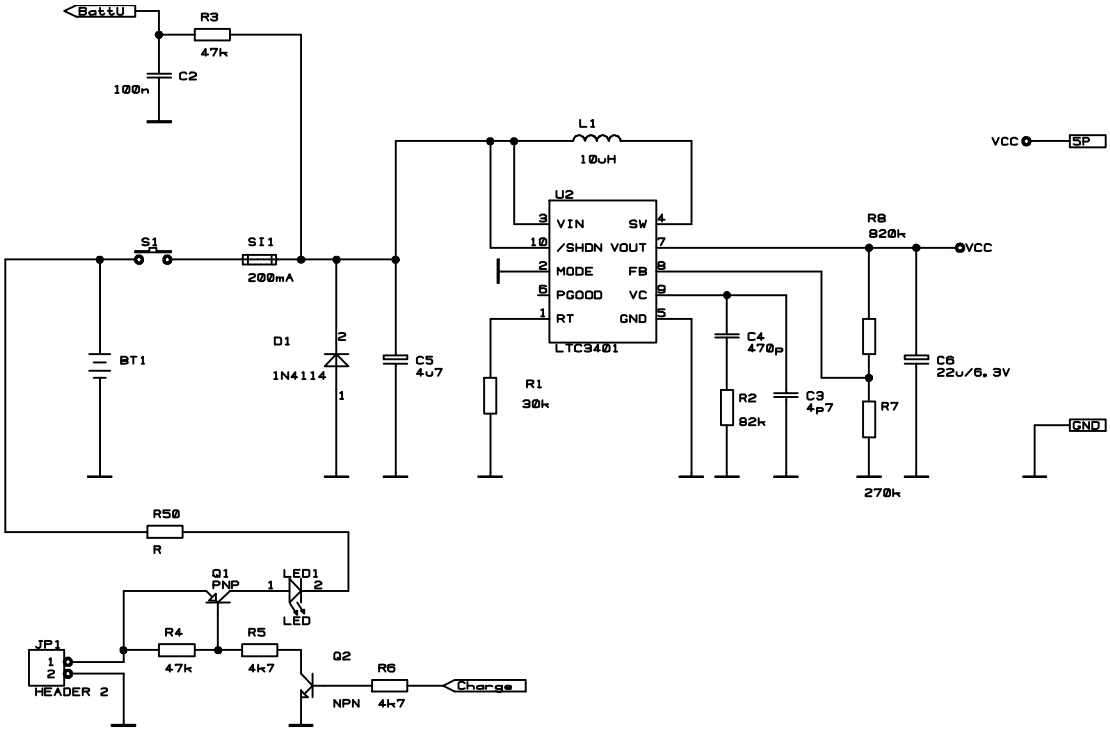



Bild 3.6 Stromversorgungseinheit der Miniaturroboter MauSI 2

Die reine Schaltungstechnik erscheint auf den ersten Blick gesehen simpel. Einfach ist jedoch nur die Schaltung selbst.

Viele wichtige Schaltungskomponenten, wie elektronischer Schalter, Freilaufdiode, Regeleinheiten, etc. sind im IC LTC3401 integriert.
Infolge der relativ hohen Schaltfrequenz von 1 MHz können nicht alle Bauelemente als konzentrierte Bauteile betrachtet werden. Insbesondere die Eigenschaften von C5, L1 und C6 im Zusammenhang mit der Anordnung (Layout) auf der Leiterplatte bestimmen maßgeblich die Funktion und die Eigenschaften des Reglers.
Die Bauteile müssen so dicht wie möglich an den Anschlüssen des LTC3401 angeordnet werden. Die Plazierung der Bauelemente und die Verdrahtung erfolgen einseitig, so daß die gegenüberliegende Leiterplattenseite komplett als Massefläche ausgeführt werden kann.

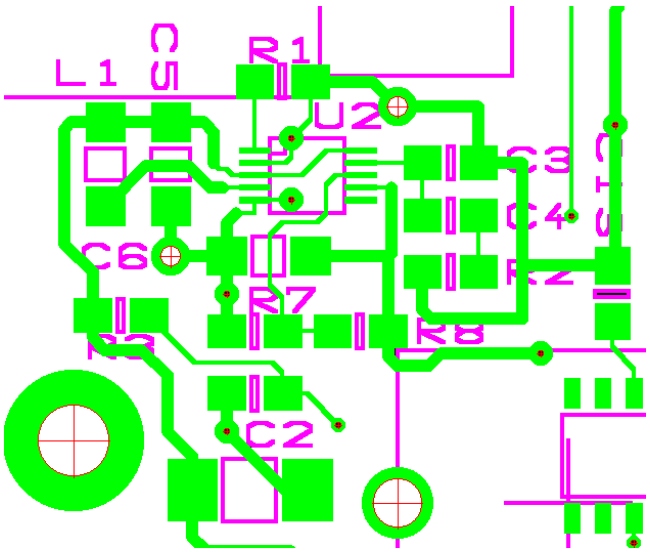



Bild 3.7 Layoutdetail des *step-up*-Reglers (reale Größe 10 x 5 mm)

Ebenso wichtig ist die Auswahl der richtigen Bauelemente. Bei Schaltfrequenzen im Megahertz-Bereich und Schaltströmen von einigen hundert Milliampere sind die üblichen Elektrolyt- oder Tantalkondensatoren ungeeignet. Innerhalb der kurzen Schaltzeiten können dann die notwendigen Impulsspitzenströme von C5 nicht geliefert werden. Der Eingangskondensator muß demzufolge extreme Forderungen bezüglich geringer Verluste in diesem Frequenzbereich erfüllen.
Besonders geeignet sind Kondensatoren der Fa. Taiyo Yuden, die spezielle Vielschicht-Keramikkondensatoren fertigt. Die Abbildung zeigt die Frequenzabhängigkeit von Impedanz und ESR (ESR = *equivalent series resistor*). Im interessierenden Frequenzbereich von 1 MHz ist die Impedanz besonders gering, so daß die notwendige Impulsleistung im Schaltmoment aus dem Kondensator entnommen werden kann.
Am Ausgang des *step-up*-Reglers ist ebenfalls ein Low-ESR Typ einzusetzen, damit die Ripple-Spannung klein bleibt.

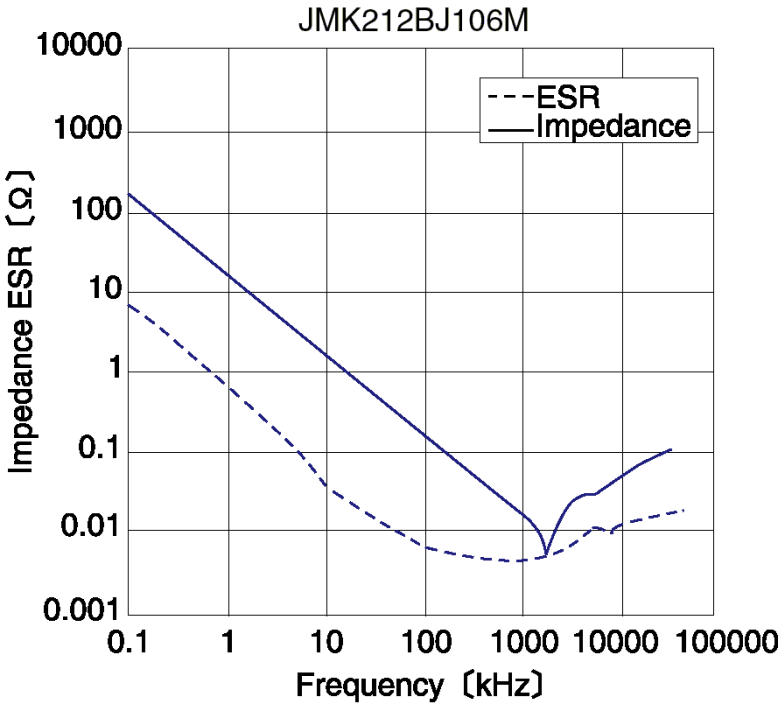



Bild 3.8 Impedanz und ESR-Verhalten in Abhängigkeit von der Frequenz

Die weiteren Schaltungskomponenten sind unkritisch. Über den Tiefpaß C2/R3 gelangt die Batterie- bzw. Akkuspannung an den A/D-Wandler des Controllers. Der Transistoren Q1/Q2 ermöglichen eine gesteuerte Ladung des Akkus, z.B. von einer Solarzelle.

### 3.3.2 Betriebsdauer des Roboters

Die Verwendung handelsüblicher LR6 Zellen gestattet eine abgestufte Betriebsdauer des Roboters für unterschiedliche Anwendungen. In dieser Zellenbauform werden sowohl Primärzellen (Akaline und Lithium) als auch Sekundärzellen (NC oder NiMH-Akkus) angeboten.

Als Standard-Stromversorgung wird ein NC bzw. NiMH Akku verwendet. In [16] werden einige Eigenschaften üblicher NC-Akkus hinsichtlich ihres Einsatzes für den Miniroboter untersucht.

Die Stromversorgung ist so ausgelegt, daß sie sowohl mit Akkus als auch Batterien zusammen funktioniert. Es sind minimal 0,5 V Eingangsspannung zum Betrieb erforderlich. Wegen des gewählten *step-up*-Reglerprinzips steigt der Eingangsstrom mit kleiner werdender Eingangsspannung. Infolge der Schaltverluste fällt dann der Wirkungsgrad der Schaltung unter 50 % bei Eingangspannungen $U_e < 1$ V.

Die Betriebszeit des Roboters ergibt sich aus folgenden Parametern:

$$t_{Aktiv} = \frac{C_{Power}}{\frac{V_{cc}}{V_i} \cdot I_{vcc}} \cdot \eta \,. \qquad (3.4)$$

Die Verwendung von NC- bzw. NiMH-Akkus ergibt sich eine Nutzungszeitdauer von ca. 3,5 h bei $C_{Power}$ = 1100 mAh, $V_{cc}$ = 5 V, $V_i$ = 2,4 V, $I_{cc}$ = 145 mA und einem Wirkungsgrad des Schaltreglers von ca. 90 %.
Alkalinebatterien oder Lithiumzellen (Hochstromzellen) steigern die Betriebsdauer nochmals recht erheblich. Hier muß jedoch der Spannungsverlauf während der Entladung stärker berücksichtigt werden. Die Formel (3.4) stimmt hier nur eingeschränkt.
Besonders beim Einsatz von Alkaline-Zellen ist eine relativ starke Abhängigkeit von Entladespannung und Entladezustand zu berücksichtigen. Dies führt im Bereich der Entladeschlußspannung zu einem erheblichen Anstieg des Entladestromes. Dieser Effekt bewirkt durch den ebenfalls beim Entladeende angestiegenen Innenwiderstandes, einen sich zunehmend beschleunigenden Leistungsverlust.
Der einzige Vorteil dieses Verhaltens besteht in der recht genauen Abschätzung der Restenergie der Zelle durch laufende Spannungsüberwachung.

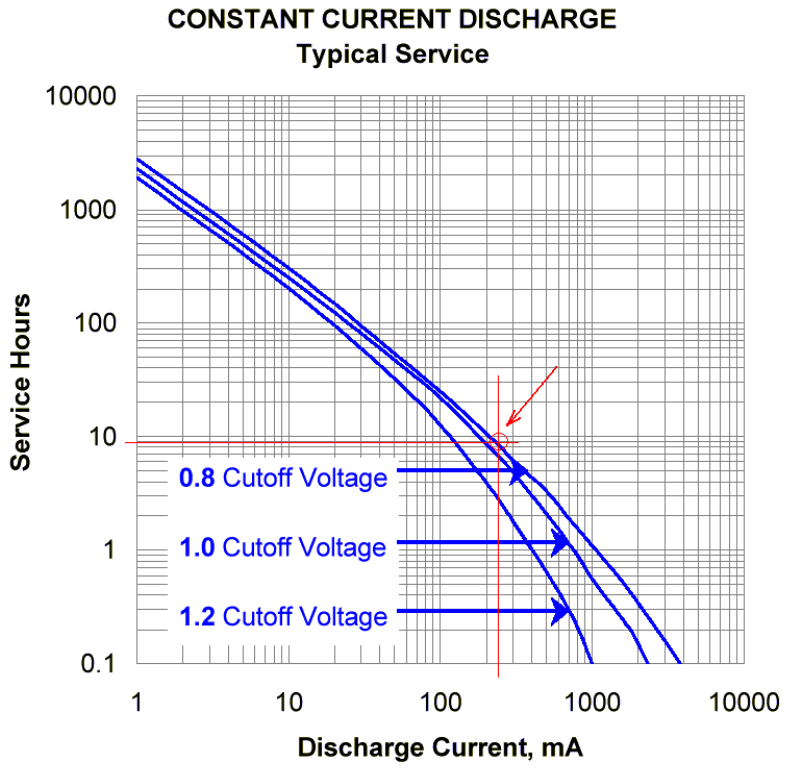



Bild 3.9 Entladekennlinie einer Alkaline-Zelle der Größe LR6

### 3.3.3 Ansteuerung und Motorregelung der DC-Motoren

Die DC-Motoren müssen vom Controller aus in beiden Drehrichtungen möglichst genau drehzahlgeregelt werden. Die einfachste Variante der Drehzahlbeeinflussung von permanent erregten DC-Motoren besteht in der Einstellung der Motorspannung. Da der Motor über die Polarität der Spannung auch in seiner Drehrichtung beeinflußt wird, erfolgt die Ansteuerung in Form einer Brückenschaltung. Ein einfaches Beispiel zeigt Abbildung 3.10.

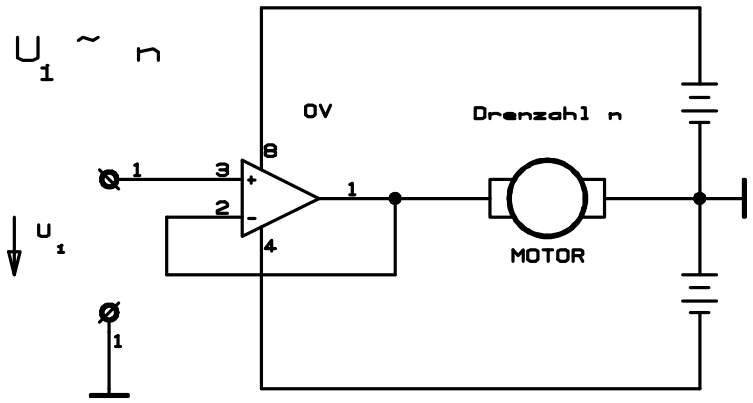



Bild 3.10 Brückenschaltung zur Drehzahleinstellung eines DC-Motors

In diesem Fall wird die Motorspannung (Ankerspannung) von einem Operationsverstärker erzeugt. Der OV ist in dieser Variante nur der Leistungstreiber.
Die Spannung $U_i$ wird z.B. von einem D/A-Wandler generiert. Die Motordrehzahl n ist der Ankerspannung proportional. Infolge der Verlustleistung, die im OV entsteht, ist dieses Verfahren ungeeignet. Interessant ist es vor allem in Bereichen, wo geringste Motorleistungen präzise gesteuert werden sollen.
Für den Roboter kommt eine getaktete Motorsteuerung zum Einsatz. DC-Motoren können mit Hilfe eines pulsweitenmodulierten Signales in ihrer Drehzahl gesteuert werden. Anstelle der linearen Ankerspannung wird ein Taktimpuls angelegt. Die Spannung des Impulses entspricht der maximalen Betriebsspannung des Motors. Die Frequenz des PWM-Signales wird so gewählt, daß die Massenträgheit des Ankers die aufeinanderfolgenden Impulse integriert, damit der Motorlauf ruckfrei wird. Die Ansteuerstufe für den Motor wird als Brückenschaltung ausgeführt, somit entfällt die Notwendigkeit einer bipolaren Spannungsversorgung. Auf einen diskreten Aufbau der Brückenschaltung aus Einzeltransistoren kann verzichtet werden, es stehen genügend integrierte Bausteine zur Verfügung.

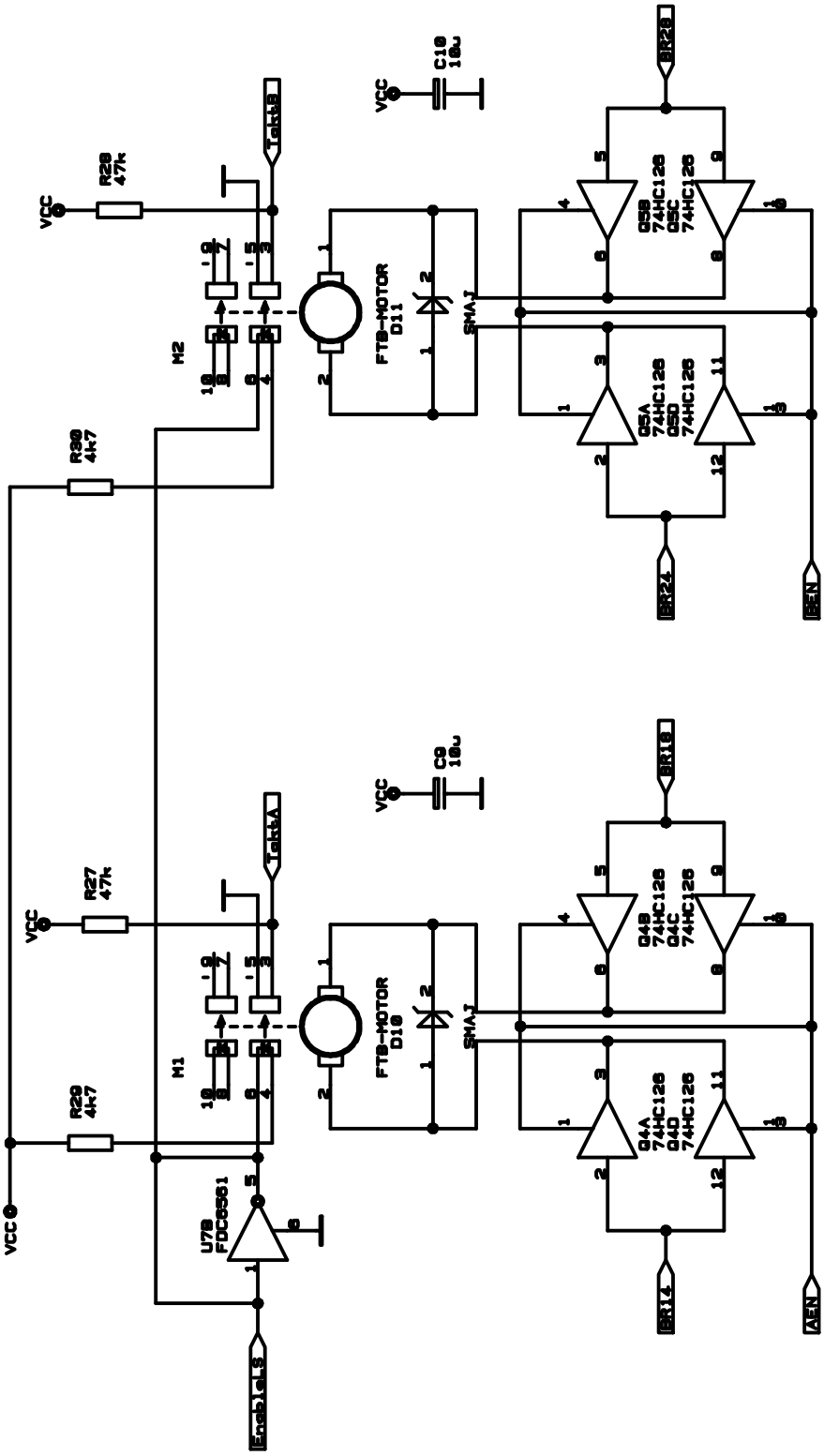



Bild 3.11 Getaktete Motoransteuerung mit optischen Encodern

Auf den Einsatz von Standard-ICs wie L293 oder L6204 (Fa. ST-Microelectronics) wird dennoch verzichtet. Versorgungsströme für die ICs von bis zu 40 mA bzw. Mindestspannungen von 12 V bescheren unnötige Verluste. Anstelle der Treiber-ICs kommen Standard-Logik-Bausteine der Serie 74ABT126 zum Einsatz, deren maximale Ausgangströme $I_{max}$ = 64 mA zur Ansteuerung der Motoren völlig ausreichend sind.
Wegen des gewählten Antriebsprinzips, *differential drive*, müssen die Motordrehzahlen geregelt werden. Zur Drehzahlerfassung dienen optische Encoder. Je eine Lichtschranke wird durch ein auf der Motorwelle angebrachtes Flügelrad bei jeder Motorumdrehung zweimal unterbrochen.

### 3.3.4 Sensoren

Wesentliche Kennzeichen von Robotern sind Sensoren zur Erfassung von Umweltreizen. Die Vielfalt möglicher Reize und die dazu erforderliche Sensorik ist beinahe unüberschaubar, die Auswahl fällt dementsprechend schwer. Zur einfachen Hinderniserkennung wird der Roboter deswegen mit einigen einfachen IR-Sensoren ausgestattet und kann darüber hinaus mit einem komplexen Kamerasensor bestückt werden. Da das System relativ offen gehalten wurde, ist auch die Verwendung weiterer Sensoren möglich. Einzige Bedingungen: geringer Energieverbrauch!

#### 3.3.4.1 Infrarot-Abstandsdetektoren

Da sich der Roboter in einer hindernisgespickten Modellumgebung zurechtfinden soll, braucht er Sensoren, die ihm Hindernisse möglichst frühzeitig melden sollen. Der einfachste Hindernisdetektor, ein durch Berühren des Hindernisses ausgelöster Schalter, ist zwar auch am Roboter vorhanden, es sollte jedoch elegantere kontaktlose Lösungen geben. Ultraschall und Radar scheiden aus Größe- und Kostengründen aus.
Optische Sensoren scheinen das Mittel der Wahl darzustellen. Sehr gute Ergebnisse in der Hindernis- und Abstandsdetektion liefern PSD-Sensoren. Diese Bauelemente senden gerichtetes Licht aus und werten den reflektierten Anteil mit einem positions-empfindlichen Element aus. Dadurch sind sowohl Aussagen über ein Vorhandensein, als auch eine gleichzeitige Abstandsschätzung möglich. Leider arbeiten die Sensoren im Nahbereich (< 10 cm) nicht mehr.

Da besonders aus diesem Entfernungsbereich Aussagen über Hindernisse und deren Entfernung benötigt werden, ist der Abstandssensor eine Kombination aus IR-Sender und IR-Empfänger.

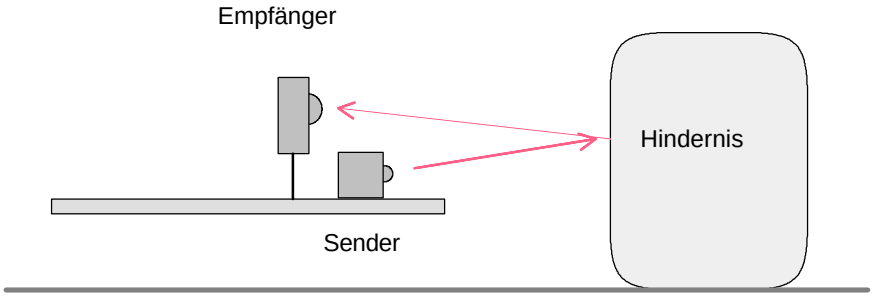
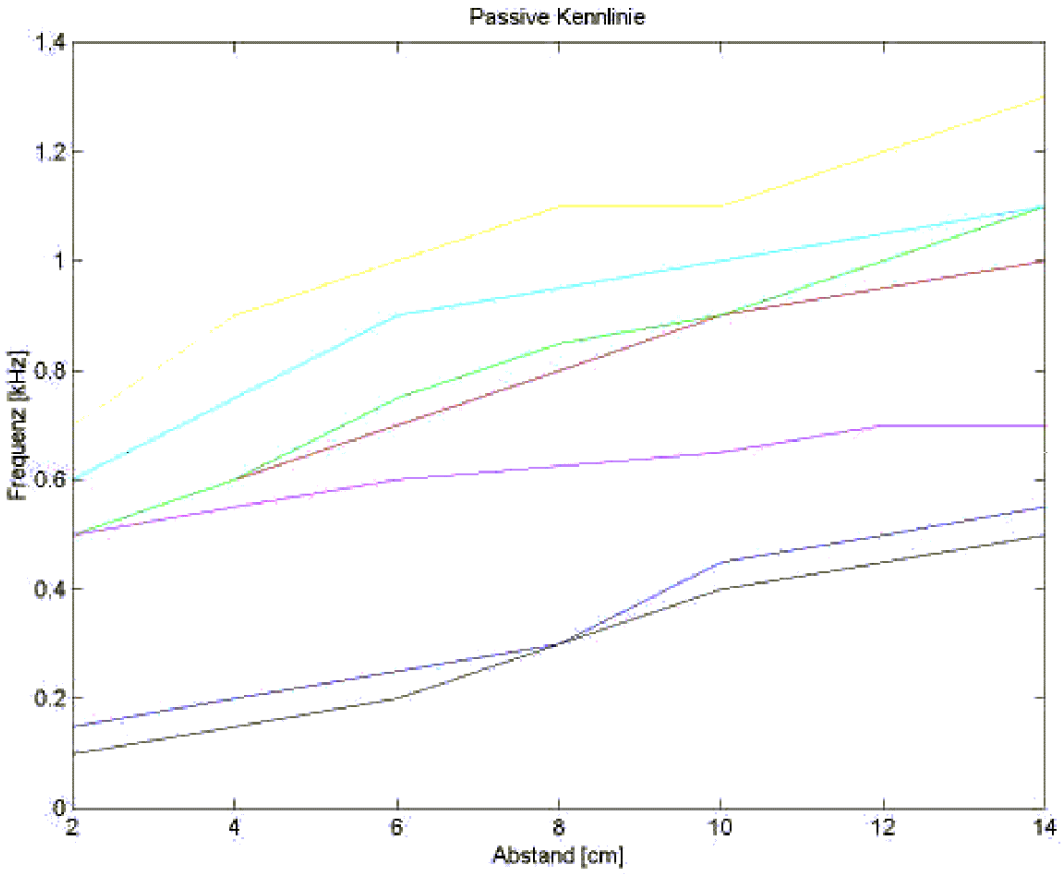



Bild 3.12 Prinzipaufbau der IR-Abstandsdetektoren

Das Verfahren ist nicht neu. In diesem Fall wird jedoch ein empfindlicher linearer Licht-Spannungs-Wandler als Empfänger verwendet. Dieser Wandler ist ein integrierter Baustein, der einen Spannungswert liefert, der direkt proportional zur empfangenen Lichtmenge ist. Die Fehler üblicher nichtlinearer Kennlinien einfacher Fototransistoren werden wirksam unterdrückt. Der Signalerfassungsbereich reicht von 0,2...80 µW/cm$^2$ (400 : 1).
Als Maß für die Entfernungsschätzung dient die Messung und Auswertung des vom Sender ausgestrahlten Infrarot-Lichtes. Zur Vermeidung von Meßwertverfälschungen besitzt der Empfänger einen Infrarot-Filter. Da dennoch mit nicht unerheblichen Störungen durch wechselnde Umgebungslichtverhältnisse gerechnet werden muß, soll eine intelligente Meßwertverarbeitung Störungen herausrechnen.
In [17] wurden dazu ausführliche Untersuchungen mit Hilfe von Licht-Frequenz-Konvertern (TSL245) unternommen. Diese Bausteine sind äquivalent zu den im Roboter verwendeten TSL260, mit dem Unterschied, daß hier eine Spannung anstelle der Frequenz ausgewertet wird. Die folgenden Kennlinien zeigen den prinzipiellen Intensitätsverlauf am Sensor (Probekörper aus grauem Pappkarton als Hindernis).
Der Sensor wird in zwei Modi betrieben, aktiv und passiv. Passiv bedeutet, daß lediglich das Umgebungslicht gemessen wird. Im Bild 3.13 ist die Abhängigkeit von Abstand und Lichtintensität dargestellt. Es wird klar, daß aus diesen Werten allein keine brauchbare Abstandsschätzung oder Hinderniserkennung möglich ist. Damit das funktioniert, wird ein zweiter Mode benötigt, der Aktiv-Mode.

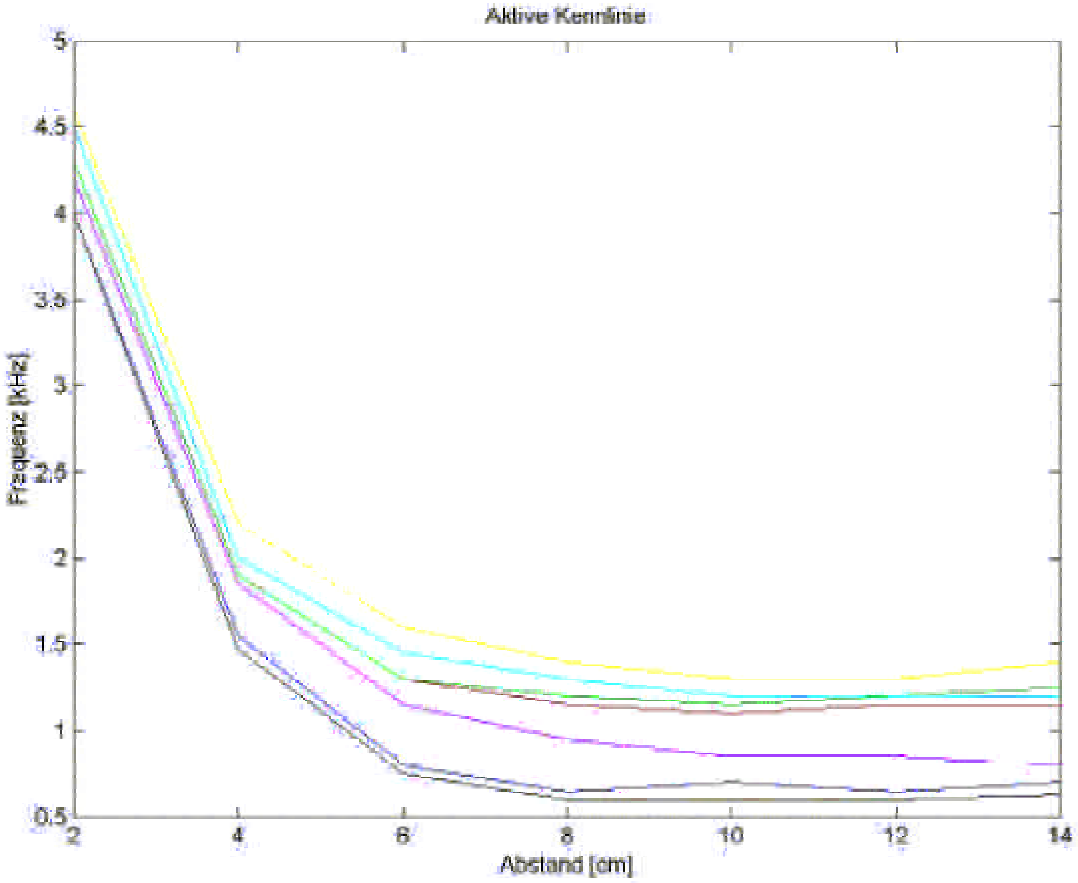



(Gelb: Sensor auf Fenster gerichtet, Halogenstrahler an, Neonlicht aus
Rot: Sensor auf Fenster gerichtet, Halogenlicht trifft Sensorfläche direkt, Neonlicht aus
Grün: Sensor auf Fenster gerichtet, Halogenlicht trifft Sensorfläche direkt, Neonlicht an
Blau: Sensor auf Fenster gerichtet, Halogenstrahler aus, Neonlicht aus
Schwarz: Sensor auf Fenster gerichtet, Halogenstrahler aus, Neonlicht aus
Magenta: Sensor vom Fenster weg gerichtet, Halogenlicht aus, Neonlicht aus
Zyan: Sensor vom Fenster weg gerichtet, Halogenstrahler an, Neonlicht aus)

Bild 3.13 Abstands-Frequenz-Kennlinie des IR-Sensors (Passiv-Mode)

Im Aktiv-Mode ergeben sich schon deutlich besser auswertbare Meßwerte bezüglich Abstand und Frequenz. Entsprechend der Umgebungshelligkeit wird dem Empfangssignal ein Offset aufgeprägt. Damit dieser Fehler erkannt und herausgerechnet werden kann, werden beide Kennlinien in Relation zueinander gesetzt. Damit scheint dann eine brauchbare Abstandsschätzung möglich zu sein.
Trotzdem werden die optischen Bauelemente zusätzlich gegen Umgebungslichteinflüsse abgeschirmt. Ein Überzug aus schwarzem Schrumpfschlauch mit entsprechenden Öffnungen zum Lichteintritt hat sich bewährt.

In der Abbildung 3.15 sind die angenährten Werte (aus beiden Kennlinien errechnet) für Frequenz und Abstand dargestellt. Die untere Kurve der Abbildung zeigt den prozentualen Fehler. Die Fehlergrenzwerte sind akzeptabel, sollte eine größere Genauigkeit gefordert werden, muß die Anzahl der Linearisierungen vergrößert werden.




6 (Gelb: Sensor auf Fenster gerichtet, Halogenstrahler an, Neonlicht aus
Rot: Sensor auf Fenster gerichtet, Halogenlicht trifft Sensorfläche direkt, Neonlicht aus
Grün: Sensor auf Fenster gerichtet, Halogenlicht trifft Sensorfläche direkt, Neonlicht an
Blau: Sensor auf Fenster gerichtet, Halogenstrahler aus, Neonlicht aus
Schwarz: Sensor auf Fenster gerichtet, Halogenstrahler aus, Neonlicht aus
Magenta: Sensor vom Fenster weg gerichtet, Halogenlicht aus, Neonlicht aus
Zyan: Sensor vom Fenster weg gerichtet, Halogenstrahler an, Neonlicht aus)

Bild 3.14 Abstands-Frequenz-Kennlinie des IR-Sensors (Aktiv-Mode)

Aus der Abbildung 3.14 ist die starke Nichtlinearität der Kurve ersichtlich. Ab Entfernungen von mehr als 8 cm ist die Frequenzänderung kaum noch als Maß für die Entfernung brauchbar.

Im Bereich zwischen 4...6 cm ist mit großen Fehlern in der Entfernungsschätzung verschiedenfarbiger Körper zu rechnen.
Für die Implementierung ist eine Linearisierung der Kurve durch zwei lineare Interpolationen vorgeschlagen (Abbildung 3.15). Eine Aproximation für Entfernungen von größer 5 cm ist nur sinnvoll, wenn innerhalb der Modellumgebung ausschließlich einfarbige Hindernisse bzw. einfarbig verkleidete Roboter Verwendung finden.
Für höhere Ansprüche, hinsichtlich sehr kleiner Rechenzeiten, können die Werte in genügend kleiner Abstufung auch direkt in einer Tabelle im ROM des Steuerkontrollers abgelegt werden.

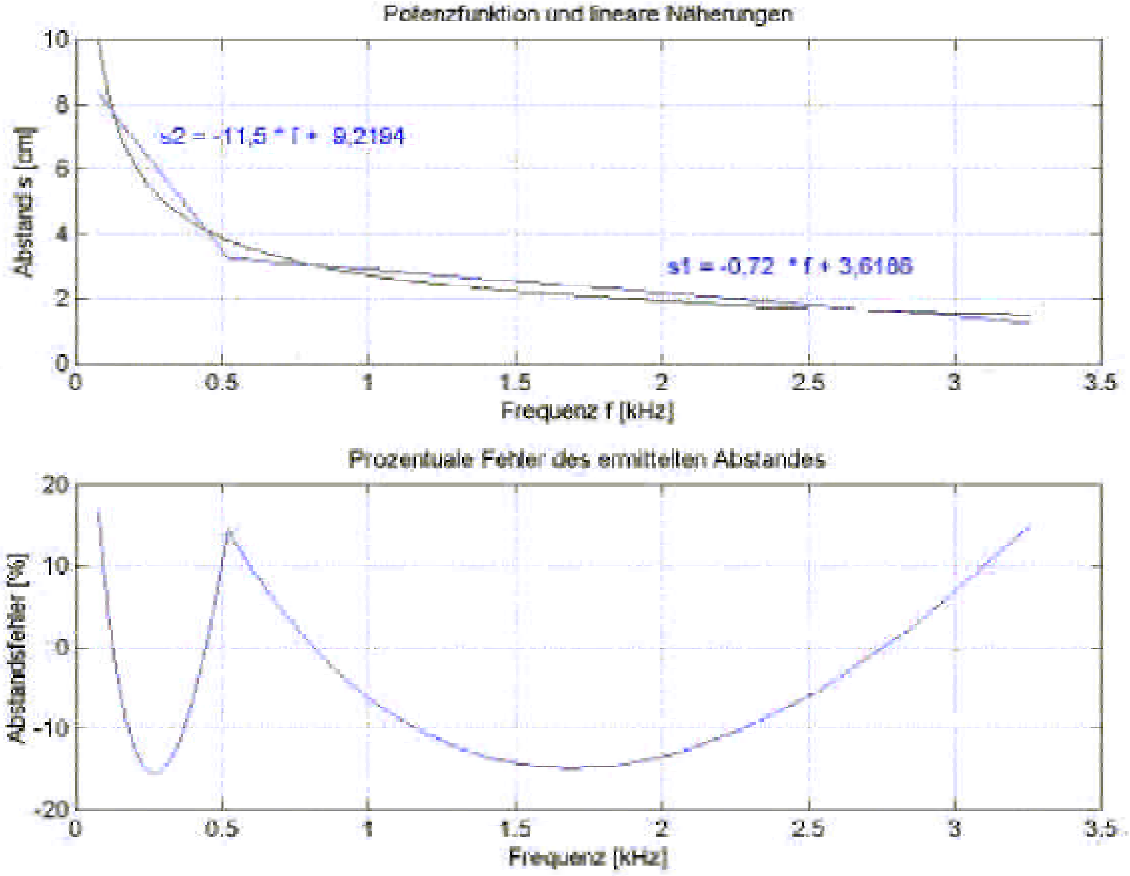



Bild 3.15 Abstands-Frequenz-Kennlinie mit prozentualem Fehler

### 3.3.4.2 Beschleunigungssensoren

Die breite Verfügbarkeit mikromechanischer Beschleunigungssensoren bietet die Möglichkeit, Meßwerte zur Lage im Raum (Kippwinkel, Rollwinkel) bzw. zu Bewegungsänderungen (Beschleunigung, Bremsung) zu gewinnen. Über kurze Zeiträume hinweg, können die Meßwerte im Zusammenhang mit anderen Meßgrößen auch Informationen zur zurückgelegten Wegstrecke bzw. zu deren Prognose liefern.
Es ist jedoch nicht möglich, allein auf der Basis von Beschleunigungswerten, Aussagen zur absolvierten Wegstrecke zu treffen. Geringe Meßfehler bzw. Ungenauigkeiten führen durch die zweifache Integration der Beschleunigung zu unbrauchbaren Ergebnissen [18].
Für den Einsatz im Roboter ist der Beschleunigungssensor ADXL202 der Fa. Analog Devices sehr gut geeignet. Der Meßbereich überstreicht +/- 2 g. Der Hersteller garantiert ein Grundrauschen der Signale unter 200 µg * $Hz^{-1}$. Damit sind Beschleunigungswerte von ca. 2 mg bei 60 Hz Bandbreite detektierbar. Die Leistungsaufnahme des Bausteins ist mit 0,6 mA ebenfalls genügend klein, um die Energieversorgung nicht übermäßig zu belasten.

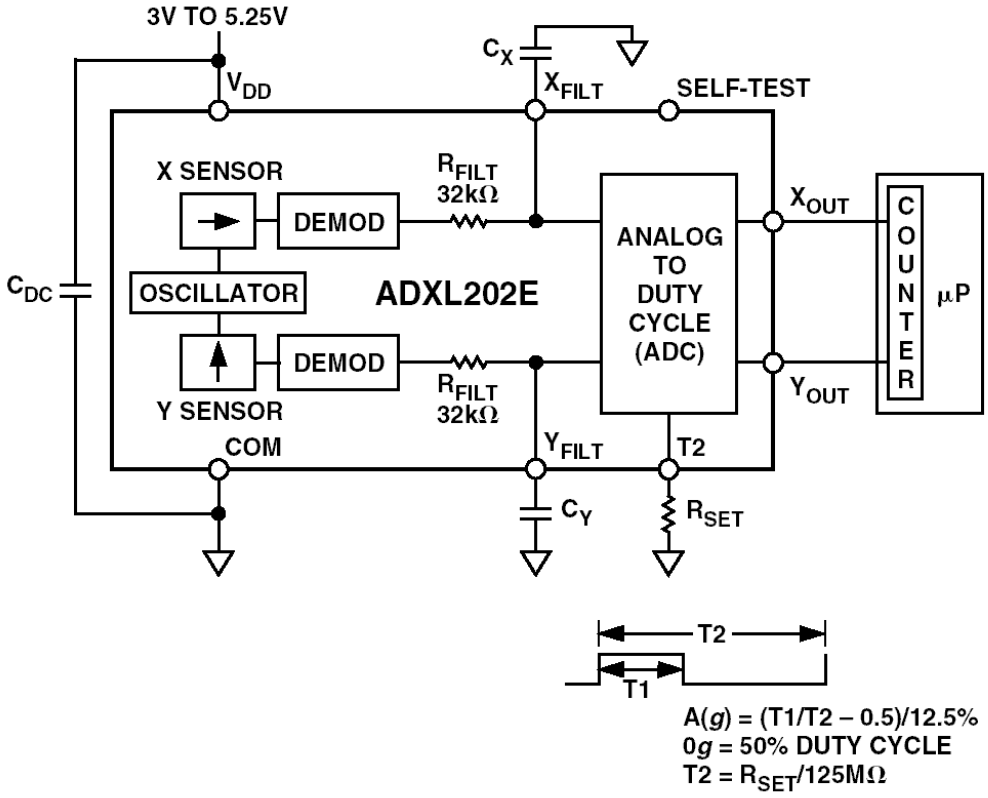



Bild 3.16 Blockschaltbild des Beschleunigungssensors

Das Funktionsprinzip des Bausteins ist an und für sich einfach. Eine Masse ist federnd aufgehängt und wird durch Beschleunigungsänderung ausgelenkt. Die Auslenkung ist zusammen mit der Federkraft ein Maß für die einwirkende Beschleunigung.
Beim ADXL202 ist die bewegte Masse als eine Kammstruktur in zweidimensionaler "Mikromechanik" ausgeführt. Diese Kammstruktur bildet einen Kondensator, dessen Kapazitätsänderung ein Maß für die mechanische Bewegung ist. Im ADXL202 sind sowohl der eigentliche Sensor als auch sämtliche Ansteuer- und Auswerteschaltungen integriert.
Die Signalausgabe erfolgt als pulsweitenmoduliertes Signal, das an entsprechenden Timereingängen des Steuercontrollers anliegt.

### 3.3.4.3 Odometrie und Ladezustandserkennung

Zur Wegstreckenerfassung sind optische Encoder an den Motorwellen der Antriebsmotoren angebracht. Der Lichtstrahl jedes Encoders wird zweimal pro Motorumdrehung durch ein Taktrad unterbrochen. Damit ergibt sich durch die Getriebeuntersetzung von 139 : 1 und dem Antriebsraddurchmesser von 42,5 mm eine Auflösung von 0,48 mm pro Takt. Diese Auflösung wird im folgenden als Schritt bezeichnet. Es wäre denkbar durch weitere Unterbrechungen die Auflösung zu erhöhen, da das Getriebespiel aber bereits in der Größenordnung der Schrittauflösung liegt, ist dies wenig sinnvoll.
Die Radencoder besitzen eine zweite Funktion. Neben der Wegstreckenmessung dienen sie zur Drehzahlregelung der Motoren.

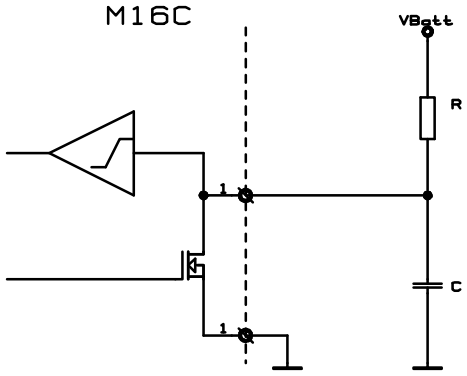



Bild 3.17 Alternative Spannungsmessung über die Lade/Entladezeit eines Kondensator

Der letzte interne Sensor ist die Ladezustandserkennung. Über einen Tiefpaß wird die Akku- bzw. Batteriespannung an einen Eingang des A/D-Wandlers geführt. Da sich aus der Akkuspannung keine exakten Aussagen zum Ladezustand treffen lassen, ist die relativ hohe Auflösung der A/D-Wandlers von 10 Bit nicht nötig. Alternativ dazu wird bei Weiterentwicklungen die Spannung über die Ladezeit eines Kondensators gemessen.
Die Eingänge des Steuercontrollers sind als Schmitt-Trigger-Stufe ausgeführt. Gleichzeitig ist per Software die Konfiguration des Eingangspins umschaltbar (Eingang, Ausgang, etc.). Zur Messung der Akkuspannung wird deswegen der Kondensator über den internen Transistor des Steuercontrollers vollständig entladen. Anschließend wird der Port-Pin auf Input umprogrammiert und ein Timer gestartet. Wird die Schaltschwelle des Schmitt-Trigger überschritten, stoppt der Timer, und ein Interrupt wird ausgelöst. Der Timerwert beinhaltet die Größe der anliegenden Akkuspannung. Zwar ist die Schaltschwelle des Schmitt-Triggers nur ungenau spezifiziert ($V_{cc}/2$, ca. 2,5 V), sie bleibt aber konstant, so daß die Auswertesoftware eine relativ genaue Trendschätzung der Versorgungsspannung vornehmen kann. Es ist also sichergestellt, daß die völlige Entladung des Akkus rechtzeitig erkannt wird.

### 3.3.5 Telemetrie

Für die Koordination mehrerer Roboter untereinander bzw. dem Datenaustausch mit einer Basisstation bzw. mit einem Operator ist eine drahtlose Datenverbindung unerläßlich.
Die Telemetrieverbindung wird über ein LPD-Gerät (LPD - *low power device*) im 433 MHz ISM-Band abgewickelt.
Um unnötige Entwicklungsarbeit zu vermeiden, wird ein fertiges OEM-Modul der Fa. Radiometrix eingesetzt. Diese Module beinhalten einen kompletten 433 MHz Transceiver in einer Baugröße von 33 x 23 x 5 mm.
Dieses Funkmodul ist für die Datenübertragung digitaler Signale optimiert. Das zu übertragende Signal wird frequenzmoduliert, d.h. die Trägerfrequenz wird entsprechend dem logischen Signalpegel umgetastet. Die Steuerpins des Moduls sind TTL/CMOS kompatibel, so daß die Kopplung an einen Mikrocontroller unproblematisch ist und nur einige Besonderheiten von Funkübertragungen berücksichtigt werden müssen.
Funkübertragungen sind prinzipiell fehlerbehaftet. Es ist nie auszuschließen, daß es zu Datenverlusten infolge von Störungen durch andere Funkdienste, schlechter Übertragungsqualität oder Störungen anderer technischer Geräte kommt. Die verwendeten

Funkmodule arbeiten alle auf der gleichen Frequenz. Gleichzeitiges Senden und Empfangen von Botschaften ist nicht möglich.

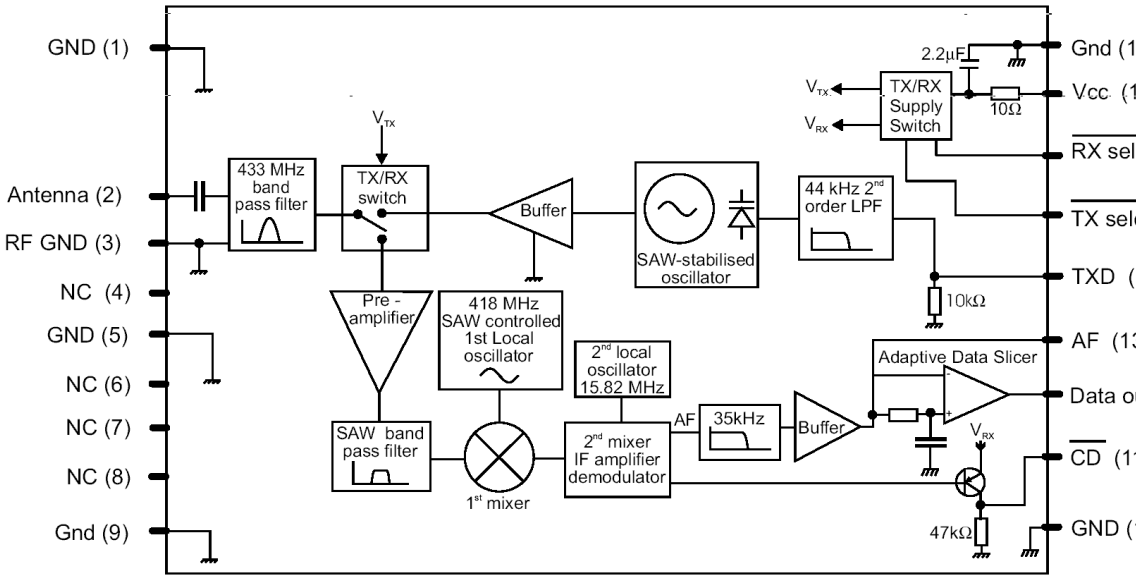



Bild 3.18 Blockschaltbild des 433 MHz - Funkmoduls

Der Aufbau einer Funkverbindung ist an bestimmte Einschalt- bzw. Einschwingzeiten gebunden. Anhand der folgenden Abbildung (Bild 3.19) werden die wesentlichen Parameter erläutert.

Es wird angenommen, daß das Modul aus dem Stand-By heraus aktiviert wird, d.h. der Empfänger ist zu Beginn stromlos.

Nach dem Zuschalten der Stromversorgung benötigt das System eine Einschwingzeit (*data setting time*) von ca. 3 ms. Innerhalb dieser Zeitspanne sind /CD (*Carrier Detect*) und RxD (*Rx Data*) ungültig. Besonders die Verknüpfung von /CD und /RX-Select ist problematisch, /CD ist auch dann aktiv, wenn der Empfänger stromlos ist.

Ein weiteres wichtiges Kriterium bei der Datenübertragung ist die Einhaltung von Maximalzeiten bei denen das Datensignal *high* oder *low* sein darf. Diese Zeitdauer, *time between code transitions*, darf 1 ms nicht überschreiten.

Die Begründung dafür ist in der Dimensionierung des *adaptive data slicers* (Bild 3.18) zu suchen. Diese Schaltstufe sorgt für die Digitalisierung der empfangenen Audiodaten. Dazu wird über einen Tiefpaß der gleitende Mittelwert des Audiosignales gebildet. Verbleibt nun das Datensignal zu lange in einem logischen Zustand, verringert sich die Spannungsdifferenz zwischen gefilterten und aktuellem Audiosignal. Kommt diese

Differenz dann in die Größenordnung der Komperatorschwelle wird der Rauschanteil des Audiosignales digitalisiert und ausgegeben (*squared noise*).

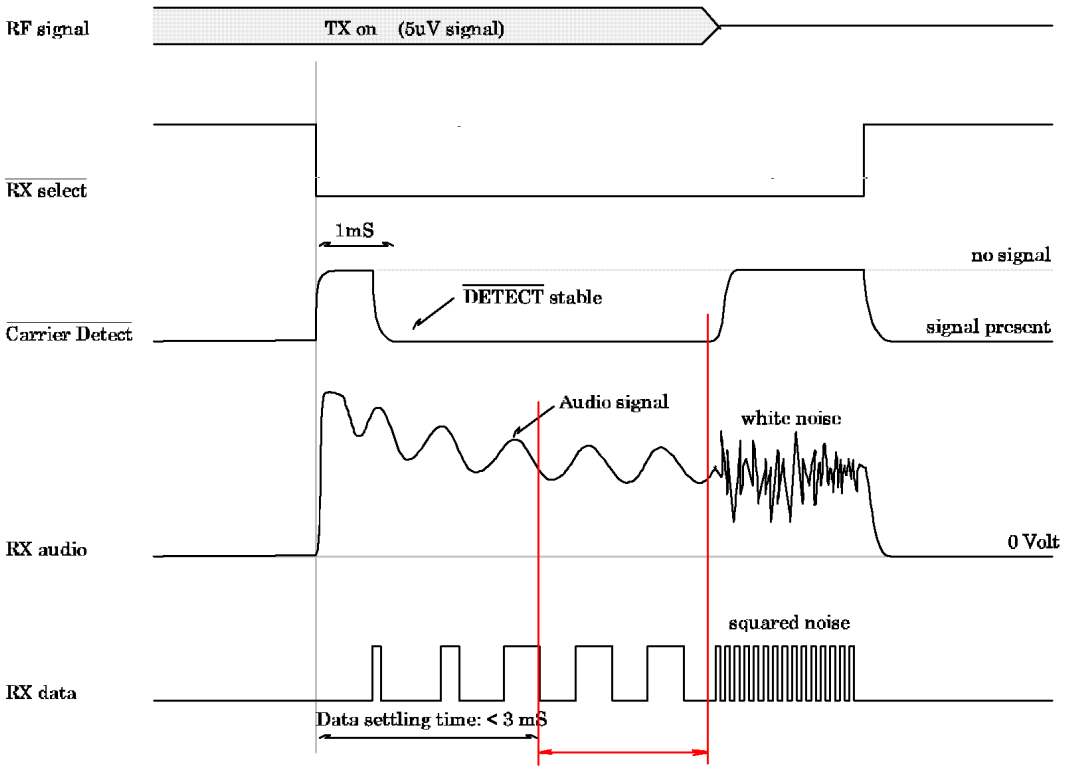



Bild 3.19 Signalspiel am Funkmodul während des Empfanges

Aus diesem Grund muß jeder Funkbotschaft ein Sync-Zeichen, 0101010... Patternmuster, vorangestellt werden. Diese Signalfolge sorgt für einen korrekten Arbeitspunkt des *adaptive data slicers*.
Letztendlich ist ungestörter Empfang nur in der Zeitspanne zwischen den Markierungen möglich. Weitere Grundlagen der digitalen Signalübertragung werden in [34] erläutert.

### 3.3.6 Der Kamerasensor

War das notwendige Equipment zur Bilddatengewinnung bis vor kurzem noch umfangreich, kostspielig und energiehungrig, so machen die besonderen Eigenschaften von CMOS-Kamera-Sensoren und der stark fallende Preis dieser Bauelemente den Einsatz von Kameras in Minirobotersystemen zunehmend interessant. Hinzu kommt die

akzeptable Leistung neuerer 16/32-Bit Mikrocontroller, so daß auch einfache Bildverarbeitung denkbar erscheint.
Für den Einsatz in den Minirobotern ist das Kameramodul OV6620 der Fa. Omnivison (*www.ovt.com*) gut geeignet. Der Baustein beinhaltet neben dem eigentlichen Bildsensor alle analogen und digitalen Baugruppen die zur Bildgewinnung nötig sind.

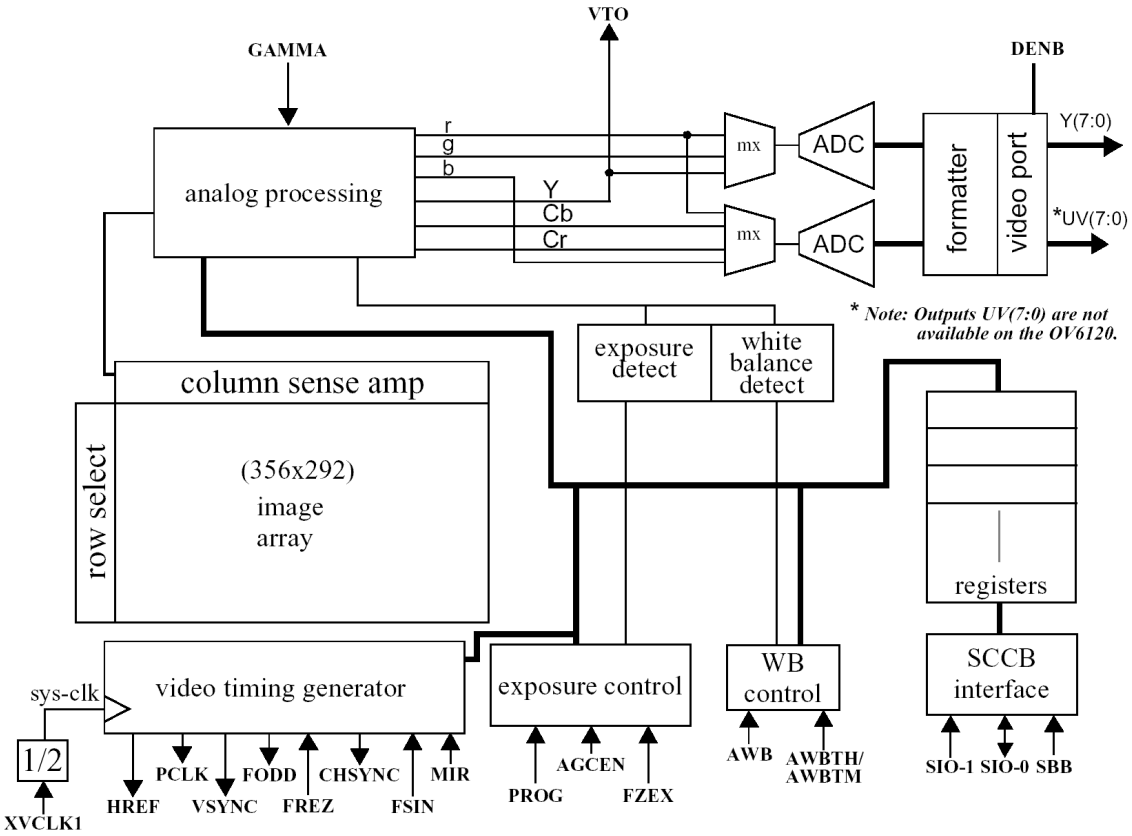



Bild 3.20 Blockschaltbild der CMOS-kamera OV6620

Der Bildsensor hat eine Auflösung von max. 356 x 292 Bildpunkten (Pixel). Über einen $I^2C$-Bus ist der Baustein vielfältig programmierbar (unterschiedliche Bildauflösungen, Abtastraten und Zeitverhalten). Nach dem Reset befindet sich der Kamerasensor im Standard-Mode. Das über eine Optik auf den Bildsensor projizierte Bild wird mit einem Pixelclock von 8,86 MHz (externer Schwingquarz 17,73 MHz) abgetastet und als Digitalwert mit 8-Bit Auflösung am Datenport bereitgestellt (Y[7:0]).
Zur Synchronisation mit dem Mikrocontroller dienen neben dem Pixelclock, VSYNC und HREF. VSYNC (*vertical sync pulse*) signalisiert den Beginn eines Bildes. HREF (*horizontal valid data output window*) kennzeichnet die gültigen Pixelclock-Impulse innerhalb einer Zeile.

Nach dem Reset werden die Bilddaten dem Controller mit einer Datenrate von ca. 8,6 MByte/s im Schwarz-Weiß-Mode bzw. mit ca. 17 MByte/s im Farb-Mode angeboten. Diese Datenrate und das damit verbundene Datenvolumen ist für den eingesetzten Controller viel zu viel. Über den $I^2C$-Bus wird deswegen der Sensor umprogrammiert. Der Pixeltakt wird auf Werte im Bereich um 1 MHz reduziert und der Bildausschnitt wird ebenfalls verkleinert. Da zu Anfang nur ein Schwarz-Weiß-Modul zur Verfügung stand, wurde mit einem Bildformat von 176 x 64 Pixel gearbeitet. Dieses Format resultiert aus dem maximal bereitstellbaren on-chip-RAM des M16C von 11264 Byte.

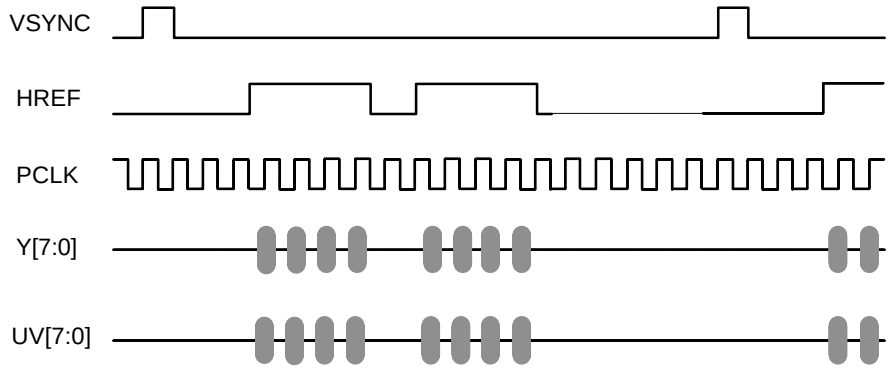



Bild 3.21 Impulsdiagramm zur Datenausgabe am Kamerasensor

Dennoch ist die Datenmenge, die in den Speicher des Controller gelangen muß, erheblich. Ein solches Datenvolumen kann nur per DMA (*direct memory access*) gehandhabt werden. Ein Auslesen per Software verschlingt zuviel Rechenleistung. Mit Hilfe des programmierbaren DMA-Controllers werden die Bilddaten mit nur einigen Prozent Prozessorlast in den RAM übertragen.

Bild 3.22 Schwarz-Weiß-Bild der CMOS-Kamera

Die Abbildung 3.22 zeigt eines der ersten per Funk übermittelten Bilder der Roboterkamera. Die Aufnahme ist relativ grob gepixelt, für Bildbetrachtungen ist dies nicht optimal, außerdem ist die Aufnahme nur in schwarz-weiß. Aus den Untersuchungen zur Bildverarbeitung wird jedoch klar, daß für maschinelle Auswertung auf der lokalen Ebene (Roboter) höhere Auflösungen nicht unbedingt zweckmäßig sind, da sonst die Verfahren zur Bildverarbeitung zu zeitintensiv werden. Eine weitere Variante des Roboters besitzt einen zusätzlichen Speicher für höher aufgelöste (352 x 288 Pixel) farbige Aufnahmen.
Die Auflösung des schwarz-weiß-Bildes in Abbildung 3.22 beträgt 120 x 80 Bildpunkte. Der Flaschenhals bei der Übertragung höher aufgelöster Bilder ist die Funkübertragung. Da der interne Speicher des Steuercontrollers zum Zwischenspeichern eines gesamten Bildes nicht ausreicht, gibt es eine Kamera mit integriertem FIFO (*first in first out memory*). Dieser Speicher ist groß und schnell genug, ein komplettes Farbbild zu puffern und kann dann später langsamer ausgelesen werden.

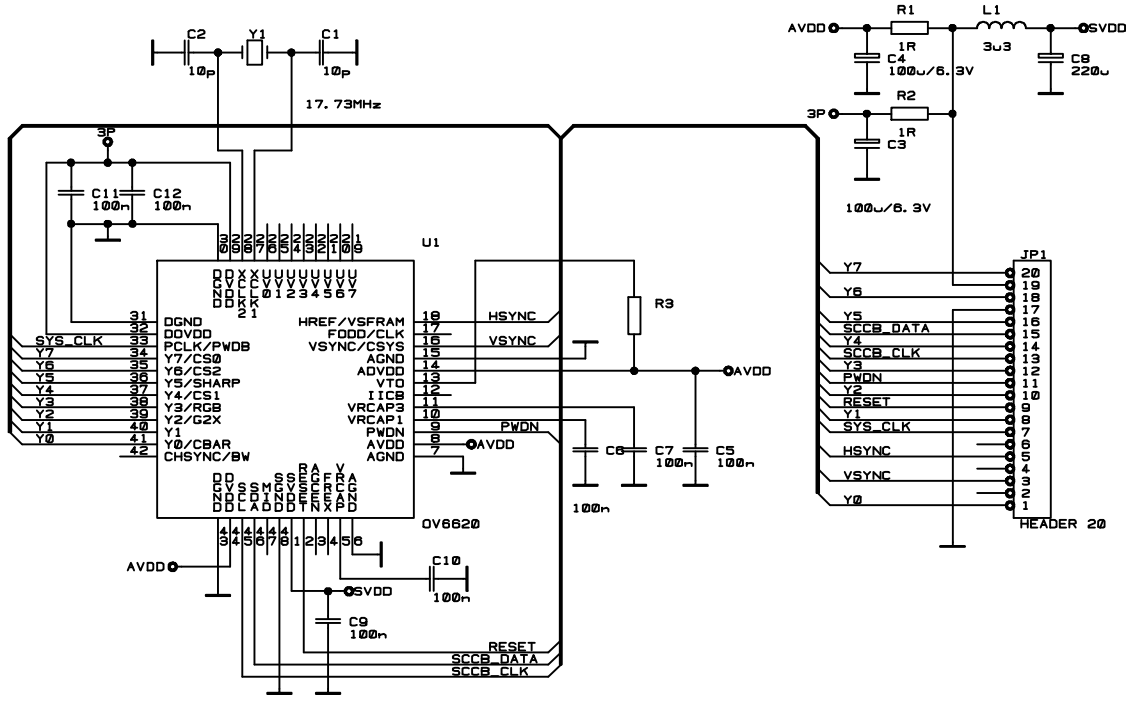



Bild 3.23 Stromlaufplan des CMOS-Sensors

# 4 Softwaredesign

## 4.1 Minimalsystem mit 8-Bit-Mikrocontrollern

Die urspüngliche Konzeption des Robotersystems stammt aus einer Veröffentlichung aus dem Jahre 1999 [19]. Die dort beschriebenen Roboter auf der Basis von 8-Bit-Controllern bildeten die Grundlage der ersten Versuche zur koordinierten Steuerung mehrerer autonomer Einheiten entweder untereinander als auch von einer Basisstation aus. Erste Ergebnisse dazu wurden in [20] vorgestellt.
Kern dieses Robotersystems waren zwei unabhängig voneinander arbeitende 8-Bit-Mikrocontroller. Der eine Controller (oben im Bild 4.1) ist für die Funkkommunikation der andere Controller (auf der unteren Leiterplatte) für Steuer- und Regelaufgaben im Roboter zuständig. Die Kopplung und Synchronisation der Controller erfolgte mittels einer seriellen Datenverbindung und eines "gemeinsamen" Speicherbereichs. Gemeinsam in diesem Sinne bedeutetet, daß ein bestimmter Teil des RAM in jedem der beiden Controller reserviert war, und dieser Speicherbereich von einer Systemfunktion auf jedem Controller laufend aktualisiert wird.

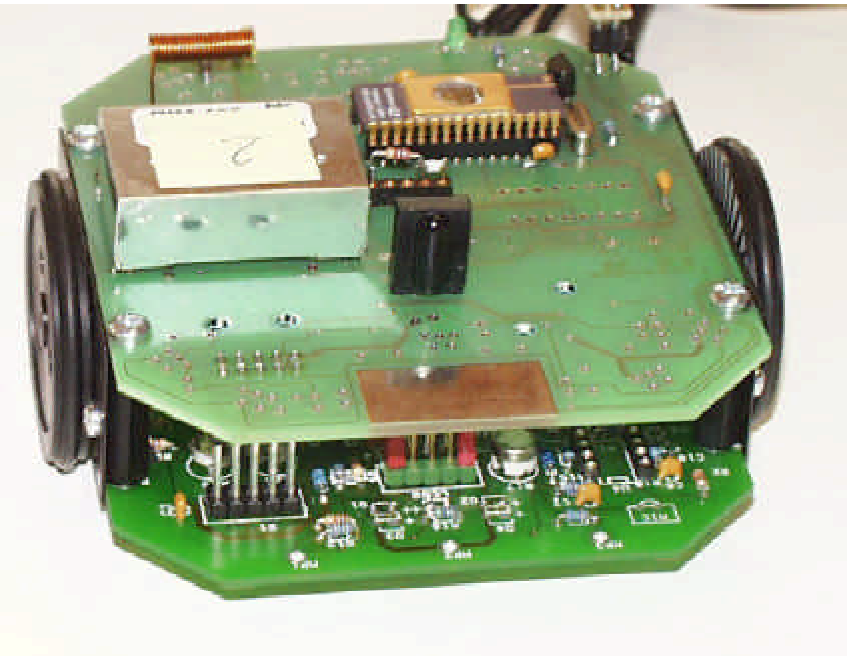


Bild 4.1 Funkrobotersystem MauSI 1

Mit diesem recht umständlich anmutenden Verfahren war die Synchronisation beider Controller sichergestellt. Letztlich kann dieser Versuch als einfaches Abbild spezifischer Multitasking-Fähigkeiten auf ein System von mehreren Rechnern gesehen werden. Trotz erheblicher Anstrengungen, das gewählte System flexibler zu gestalten und als echtes Mehrcontrollersystem zu etablieren, erwies sich das Verfahren als Sackgasse. Ohne eine schnelle Schnittstelle zur Interprozessorkommunkation, die am Prozessor vorhandene SPI (*synchron peripherial interface*) war ungeeignet, sind die Leistungssteigerungen mit dem Einsatz weiterer Controller durch den zusätzlichen Protokollaufwand in der Software mehr als kompensiert worden.
Die Experimente sind dennoch kein kompletter Fehlschlag geworden, da zum einen grundlegende Steuerprinzipien auf ihre Brauchbarkeit geprüft werden konnten. Hier hat der extreme Zwang zu optimalen Softwarelösungen in der Hochsprache C für genügend Spielraum gesorgt, diese Module in deutlich leistungsfähiger Umgebung (16-Bit-System) weiterzuverwenden.


Bild 4.2 Bildschirmansicht der Webseite zur Robotersteuerung per Internet

Zum anderen brachte ein eineinhalbjähriger Dauerversuch wichtige Ergebnisse zu notwendigen Erweiterungen für die Funkkomunikation und die Energieversorgung.
In diesem Versuch wurden mehrere Roboter in einer einfachen Modellumgebung plaziert, die von einer Web-Cam beobachtet wurde. An zwei definierten Punkten dieser

Modellumgebung waren zwei Akkuladestationen angebracht. Die Aufgabe der Roboter bestand nun darin, diese zu suchen, wenn der interne Ladezustand der Akkus unterhalb eines Schwellwertes gefallen war.
Neben der reinen Beobachtung der Modellumgebung durch die Kamera, war diese Kamera über einen Server als Webcam im Internet verfügbar. Besucher dieser Webseite konnen über Steuerbuttons die Roboter in Bewegung versetzten und durch die Modellumgebung steuern.
Der einfach gehaltene Zustandsrechner der Roboter verhinderte Kollisionen der Roboter mit der Begrenzung des "Spielfeldes" oder andern Robotern der Modellumgebung. Es konnten bis zu vier Roboter gleichzeitig kontrolliert werden.
Eine statistische Auswertung des Roboters mit der längsten Betriebszeit, ergab für diesen eine Betriebsdauer von ca. 7000 h. Von dieser Zeit war der Roboter etwa 5 % aktiv, d.h. er bewegte sich entweder auf der Suche nach der Ladestation oder er wurde von einem Besucher der Webseite gesteuert. Da die Roboter sich mit einer Geschwindigkeit von ca. 30 mm/s fortbewegen, absolvierte dieser Roboter eine Gesamtstrecke von ca. 37 km(!).
Ein weiteres Ziel dieses Aufbaus war der Test, in welcher Weise sich solch eine Demonstration zur Steigerung der Attraktivität einer Webseite eignen würde. Ohne aufwendige Werbeaktionen stieg die Zahl der monatlichen Zugriffe von ca. 50 Aufrufen vor der Demonstration auf bis zu 800 Aufrufen nachdem die Roboter aktiv waren.

## 4.2 Systemdesign mit 16-Bit-Controllern

### 4.2.1 Auswahl des Steuercontrollers

Aus dem Ergebnis der Vorversuche mit Robotern ergab sich die Forderung nach einem deutlich leistungsfähigeren Steuercontroller. Als Minimalforderungen wurden folgende Parameter aufgestellt: 64KB ROM, 10 KB RAM, 4 Timer, 2 Schnittstellen, A/D-Wandler, eine ausreichende Anzahl von programmierbaren I/O-Ports, sowie eine 16-Bit CPU. Eine weitere wichtige Randbedingung ist die Verfügbarkeit und der Preis der erforderlichen Entwicklungstools (Compiler, Debugger, etc.).
Aus der Vielzahl der möglichen Bausteine wurden Controller aus der HC12-Familie (Motorola), ST10 (ST-Microelectronics), 80C166 (Infinieon) und M16C (Mitsubishi) auf ihre Eignung untersucht.
Zwei Hauptgründe führten dann zur Auswahl der M16C Familie: leistungsfähige Entwicklungstools mit on-chip-Debugging und die Möglichkeit des späteren Einsatzes von pinkompatiblen Bausteinen mit einer 32-Bit-CPU.

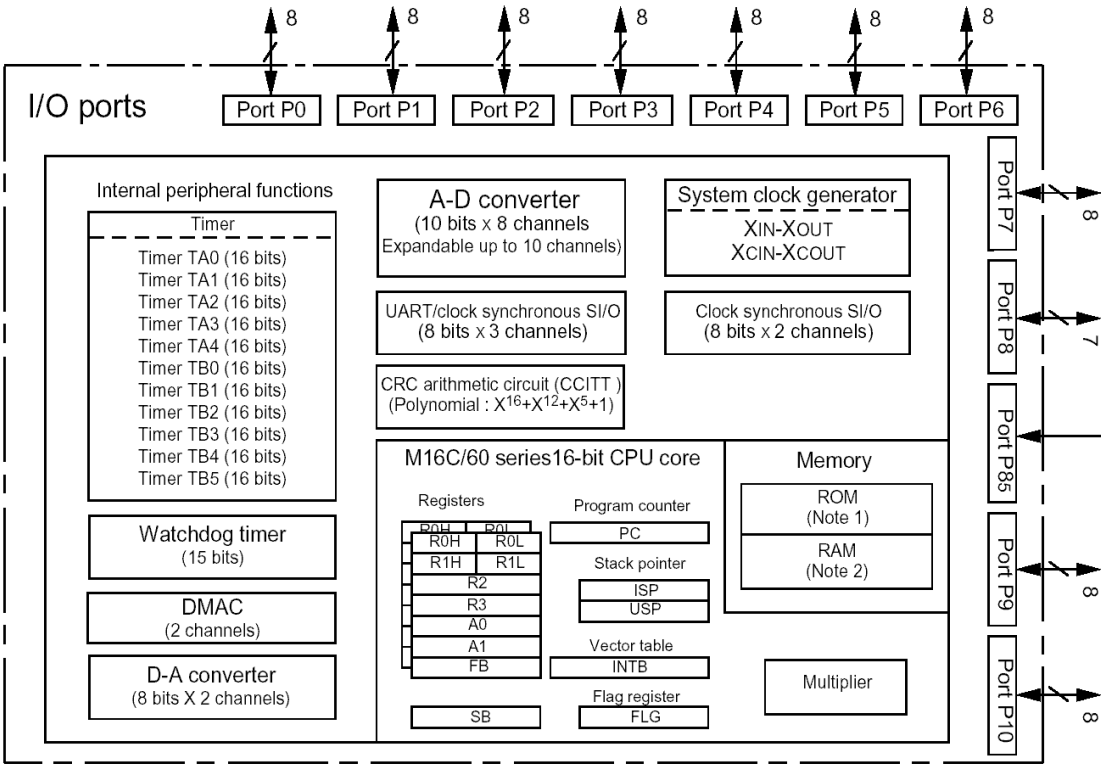



Bild 4.3 Blockstruktur des M16C-Mikrocontrollers

In der Abbildung 4.3 sind die verschiedenen internen Peripherieeinheiten dargestellt. Außer der UART1, die zum Debuggen benötigt wird, stehen für den Roboter alle anderen Einheiten zur Verfügung.

### 4.2.2 Modulares Softwarekonzept

Bevor einige wesentliche Softwarekomponenten näher erläutert werden, ist die Problematik des prinzipiellen Softwareaufbaus zu beantworten. So stellt sich zunächst die Frage, wieviel Software kann nachgenutzt bzw. wie hoch ist der notwendige selbst zu entwickelnde Anteil?
Relativ klar ist, daß spezifische Ansteuermodule, z.B. PID-Regler, selbst implementiert werden müssen. Hier auf gängige Softwareprodukte zurückzugreifen, ist nicht sinnvoll. Genauso sollten die Softwaremodule in hardwarenahe Module und höher abstrahierende Teile gesplittet werden. Letztlich soll nicht nur ein Programmierer mit dem Roboter arbeiten, sondern das System so durchschaubar bleiben, daß Ergänzungen oder Erweiterungen später problemarm durchgeführt werden können.

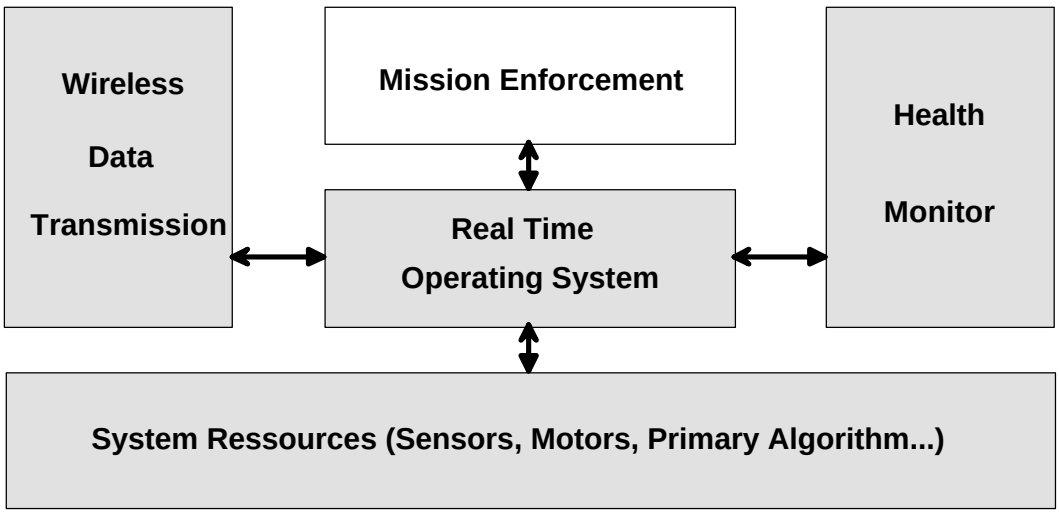



Bild 4.4 Blockstruktur des Softwareaufbaues

Aus der Abbildung wird deutlich, daß dem RTOS (*real time operating system*) eine zentrale Rolle bei der Zuteilung der CPU-Ressourcen obliegt.
Der Aufbau der einzelnen Softwaremodule wird damit klar. Jedes Modul hat einen klar definierten Funktionsumfang und beinhaltet entsprechende Softwareschnittstellen, damit andere Module die Funktionalität nutzen können. Es wird großer Wert auf die Kapselung der Module gelegt, d.h. nach Möglichkeit werden die Schnittstellen zu anderen Modulen

nur über Funktionsrufe bzw. Callback-Funktionen definiert. Es wird auch versucht einfache Prinzipien objektorientierter Programmierung zu berücksichtigen.

### 4.2.3 Echtzeitorientiertes Multitasking

Etwas anders sieht die Lage bei dem einzusetzenden Betriebssystem aus. Auch hier stellt sich die Frage: Kauft man ein System, oder baut man selbst eines? In der Industrie werden eine Reihe leistungsfähiger Systeme angeboten. Systeme wie QNX™ (*www.qnx.com*) oder CMX-Tiny+ (*www.cmx.com*) sind dazu einige Beispiele. Im Automotive-Bereich sind Systeme auf der Basis von OSEK stark verbreitet.
Allen diesen Produkten wird eine entsprechende Leistungsfähigkeit bescheinigt. Features wie z.B. *control tasks, control events, true preemption, cooperative scheduling allowed, fast context switch times* oder *low interrupt latency* sind Bestandteil des Funktionsumfangs. Hinzu kommen noch Grafikbibliotheken oder TCP/IP-Protokoll Stacks die ebenfalls angeboten werden.
Diese Eigenschaften haben dann natürlich auch ihren Preis. Je nach System werden einmalige Kosten oder Lizenzgebüren (*royalties*) pro produziertem Gerät erhoben. Für den Einsatz in der Forschung bzw. Ausbildung gibt es teilweise Sonderpreise, bzw. der Einsatz von µCOS ist in dieser Sparte kostenlos möglich.
Neben den Kosten ist zum Teil aber auch der Funktionsumfang der Systeme so groß, daß entweder erhebliche Ressourcen (RAM bzw. ROM) belegt werden oder der Einstieg in die komplexen Systeme so kompliziert ist, so daß der eine oder andere Student damit schlichtweg überfordert wird.
Der Einsatz des RTOS im Roboter benötigt auch längst nicht alle Funktionalitäten kommerzieller Systeme. Unter Zugrundelegung des geplanten Einsatzbereiches werden an eine RTOS folgende Forderungen gestellt:

- starten, verwalten und abbrechen von Tasks
- zeitgenauer Aufruf von Funktionen (Echtzeittasks)
- Möglichkeit zur Priorisierung der Rechenleistung
- Debug-Unterstützung
- minimaler Verbrauch von Ressourcen, RAM, ROM, CPU-Zeit

Auf den Gebrauch von *Messages*, vollständig präemptiver Taskverwaltung und Interrupttasks kann verzichtet werden. Die Verwendung externer Biliotheken (Grafik, TCP/IP,...) wird zum gegenwärtigen Stand ebenfalls nicht als notwendig erachtet.
In [21] wird ein Konzept zu einem minimalistischen Multitasking-Kernel beschrieben, welches viele der geforderten Eigenschaften für das Roboter-RTOS aufweist. Das dort beschriebene Multitasking-System ist für ein einfaches kooperatives Multitasking ausgelegt. Jedem Task wird eine spezifische Laufzeit zugemessen, nach der er unterbrochen und der nächste Task aus der Taskliste gestartet wird. Alle Tasks besitzen die gleiche Priorität, kein Task kann einen anderen vor Ablauf dessen Zeitscheibe unterbrechen. Das Systeme stellt folgende Funktionalität bereit:

- *UEXC_CreateTask()* - Task starten
- *UEXC_KillTask()* - Task abbrechen
- *UEXC_HogProcessor()* - Rechenzeit zur Tasklaufzeit anfordern

Diese drei Funktionen scheinen auf den ersten Blick keine besondere Leistungsfähigkeit zu versprechen. Ein einfaches System ist zwar akzeptabel, wenn jedoch aus einfach primitiv wird, kann das das gesamte Softwarekonzept zu Fall bringen.
Bevor dieser Punkt näher untersucht wird, soll das Konzept von UEXC kurz beschrieben werden. Der Schlüssel zum Verständnis liegt im Aufbau des Task-Control-Blocks (TCB). In dieser Datenstruktur sind die wesentlichen Eigenschaften festgelegt:

```
typedef struct TaskControlBlock
    {
    struct  TaskControlBlock *next;
    unsigned char tid;                  /* Task ID */
    unsigned char state;                /* Taskzustand */
    unsigned char ticks;                /* Anzahl der Zei|scheiben */
    unsigned char current_ticks;        /* verbleibende Zeitscheiben */
    void    (*func)(void);              /* Zeiger auf Task */
    unsigned char   *stack_start;       /* Stack-Beginn */
    unsigned char   *stack_end;                 /* Stack-Ende */
    unsigned char   *sp;                /* Stackpointer */
    } TaskControlBlock;
```

Die Hauptfunktion eines Multitasking-Systems ist die Speicherverwaltung und Zeitscheibenverteilung der CPU-Zeit an die jeweiligen Tasks. Da jeder Task unabhängig von allen anderen Tasks arbeitet, hat er seinen eigenen Stackbereich, der beim Taskwechsel umgeschaltet wird (**sp*,...). Die Zeitscheibenverwaltung erfolgt über Timer-Ticks. Der Scheduler wird zyklisch aufgerufen und entscheidet anhand der verstrichenen *ticks* ob eine Taskumschaltung erfolgt.

Durch die rekursive Verknüpfung der Task-Control-Blöcke in der Liste der eingetragenen Tasks ist dann die Zuordnung und der Start des folgenden Tasks geregelt. Das funktioniert nach den Prinzip des *round-robbin,* also reihum.
Mit Hilfe der Zeitscheibenzuweisung ist eine Priorisierung der Tasks untereinander möglich, derjenige mit der höchsten Priorität bekommt anteilig die meiste Rechenzeit.
UEXC unterscheidet nicht zwischen "normalen" und "Interrupt-Task". Wird in der Interruptservice-Routine anstelle der direkten Bearbeitung des Interrupts ein Task gestartet, so reiht sich dieser in die Liste der auszuführenden Tasks ein. Wann ein solcher Task wirklich ausgeführt wird, ist ein sehr schwer kalkulierbarer Zeitpunkt. Beträgt der Timer-Tick z.B. eine Millisekunde, und es wären zwei Tasks mit je 5 ms Laufzeit vor dem neuen Task in der Liste, wäre eine Verzögerung von 10 ms die Folge. Da aber ein Task zur Laufzeit neue Zeit anfordern kann, z.B. in Abhängigkeit eines Sensorsignales, besitzen diese 10 ms allenfalls statistischen Charakter.

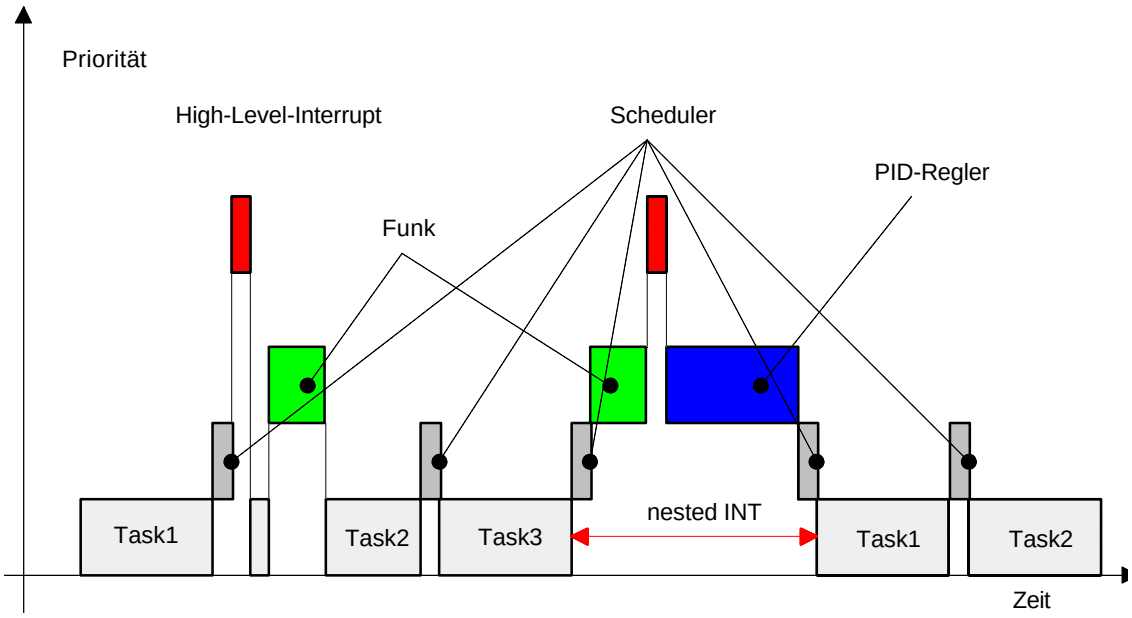



Bild 4.5 Taskumschaltung und Interrupthandling

Dieser Zeit-Jitter in der Bearbeitung zeitkritischer Programmteile wäre das k.o.-Kriterium, das den Einsatz von UEXC verbieten würde; er muß also umgangen werden.
Ein weiteres Problem sind die Debugging-Eigenschaften von RTOS und on-chip-Debugging des M16C. Der UART1-Kanal wird als Debugschnittstelle zum PC benutzt. Der interne Systemmonitor erlaubt *nested-interrupts.* Das bedeutet, daß neue Interrupts zugelassen werden, obwohl der Monitor noch aktiv ist. Im Interrupt darf aber kein

Scheduling erfolgen, sonst verabschiedet sich das System. Auch hier muß deshalb modifiziert werden.
Es ist zuerst zu untersuchen, welche Programmodule harten Echtzeitanforderungen genügen müssen und welche nicht. Zu den Kandidaten mit harten Echtzeitanforderungen zählen die Motorregler und die Funkkommunikation. Die PID-Regler der Motoren sind als zeitdiskrete Regler ausgelegt und benötigen deswegen ein starres Zeitregime. Ähnlich ist die Lage bei den Funkmodulen, hier sind auch die bereits bei der Hardwarebeschreibung erläuterten Zeitbedingung für die korrekte Funktion zu berücksichtigen. Wichtig ist natürlich auch der periodische Aufruf des Schedulers für die Taskumschaltung. Die Abbildung 4.5 verdeutlicht die Zusammenhänge. Der Scheduler wird zyklisch aufgerufen und schaltet die Tasks um. Programmteile, die harten Echtzeitanforderungen genügen müssen, werden als Interrupt-Funktion angelegt. Deren Priorität ist höher als der Scheduler-Interrupt. Ein Interrupt kann nur durch einen höher priorisierten Interrupt und nur wenn das Interrupt-Enable-Flag gesetzt ist, erneut unterbrochen werden. Bei diesem mit *nested interrupt* bezeichneten Verfahren muß auf eine entsprechende Stackgröße geachtet werden.
In der Implementierung wird das Multitaskingsystem mit zwei Timerinterrupts erweitert. Der niedrig priorisierte Timer steuert das Scheduling und der höher priorisierte Timer steuert die Echtzeitfunktionen. Damit der Anwender diese Mechanismen leicht nutzen kann, existiert für die Echtzeitfunktionen eine Funktionspointerliste in der er nur seine Applikation einzutragen braucht (siehe Beispiel).

```
/* Liste mit Echtzeitfunktionen */

const CallBackFunc TimerA0[] = { vMeineFunc          /* Applikation */
                       , n10msTick           /*(n10msTick + 1) x 10 ms */
                       , WRC_vCheckCarrier   /* Funkgerät testen */
                       , 0x04                /* alle 50 ms einschalten */
                       , vMainTimer          /* Timereinheit * /
                       , 99                  /* Aufruf pro Sekunde */
                       };
/* End of List */
```

Die Anwendung ist denkbar einfach, die Applikationsfunktion wird in die Liste eingetragen und es wird angegeben, zu welchen Zeitpunkten (Timer-Ticks) der Aufruf erfolgen soll. Die Timer-Ticks werden zurückgezählt und bei Nulldurchlauf die zugehörige Funktion gestartet. Beim Roboter beträgt der Timer-Tick 10 ms.
Durch dieses Vorgehen ist jetzt auch der Zeit-Jitter genau definierbar. Da alle n Ticks die Funktion gestartet wird, müssen die Laufzeiten der zum gleichen Zeitpunkt aktivierten Funktionen Berücksichtigung finden. Kritisch wird es an solchen Stellen, wo mehrere (oder alle) Funktionen aus der Liste "dran" sind. Das läßt sich minimieren, wenn es

möglich ist die einzelnen Zeitspannen so zu wählen, daß sie einen möglichst großen gemeinsamen Nenner bilden.
Eingangs wurde im Zusammenhang mit Multitasking auch eine geeignete Debugunterstützung gefordert. Beim M16C ist das on-chip-Debugging sehr eng mit der Anpassung des UEXC an den Prozessor geknüpft. Da diese Verknüpfung von grundlegender Bedeutung für die sichere Funktion und die Stabilität des Softwaresystems ist, soll diese Implementierung genauer beschrieben werden.
In Multitasking-Systemen wird jedem Task ein eigener Stack-Bereich zugewiesen. Dabei handelt es sich um einen reservierten Speicherblock im RAM. Die Größe richtet sich nach der Anzahl der lokalen Variablen des Tasks. Der Bereich muß im Minimum so groß sein, daß alle höherprioren Interrupts ihre lokalen Variablen zeitweise mit in diesen Bereich legen können.
Zur besseren Erläuterung wird nun angenommen, daß das System korrekt initialisiert wurde und alle Tasks laufen. Die Taskumschaltung (*context switch*) erfolgt im Interrupt durch den Scheduler *UEXC_Schedule()*. Da keine neuen Tasks vorhanden sind, braucht nur der Stackpointer "verschoben" zu werden. Dies erfolgt mit *UEXC_Resume()*.

```
void UEXC_Resume(void){
#pragma ASM
; /* _current_sp has current_task->sp restore the stack pointer */
    LDC     _uexc_current_sp, SP
    POPM    R0,R1,R2,R3,A0,A1,FB
    REIT                        ; /* Task switch */
    NOP
#pragma ENDASM
    }
```

Da C keine direkten Manipulationen am Stackpointer zuläßt, wird die Funktion in Assembler geschrieben. Im Stackbereich des Tasks, der von *active* zu *pending* geschaltet wird, hat der Interrupt einen *interrupt stack frame* errichtet, d.h. bestimmte CPU-Register, das Flag-Register und Programmzähler (PC) mit der Returnadresse sind gesichert worden. Genauso einen Stackframe findet der Assemblerbefehl REIT (return from interrupt) nun vor. Der Stackpointer zeigt mittlerweile in den Stackbereich des nun aktiven Tasks, restauriert dessen letzten Zustand der CPU-Register und dieser arbeitet an der Stelle weiter, wo er beim letzten Taskwechsel gestoppt wurde.
Der Task-Switch ist demzufolge relativ einfach zu implementieren. Komplizierter wird es, wenn ein Task generiert wird.
Die Funktion *UEXC_StartNewTask()* hat deswegen die Aufgabe, den erforderlichen Stack-Frame im reservierten Stackbereich des zu startenden Tasks anzulegen.

```
/* Start a new task */
void UEXC_StartNewTask(void){
#pragma ASM
```

```
    LDC     _uexc_current_sp,SP     ; /* load sp with new task address */

;     /* make KillSelf as the "return pc" of a new task, so if it ever
returns*/
    PUSH.b  #(_UEXC_KillSelf >> 16) ;
    PUSH.w  #(_UEXC_KillSelf & 0ffffh)
;   /* new task stack frame:  */
;   /* FLG_h, PC_h            */
;   /* FLG,PC_m  PC_l         */
;   /* puzzle FLG and PC togehter */
    STC     FLG,R0                  ; /* FLG holen */
    AND.w   #00ffh,R0               ; /* 0000000011111111 Maske aufsetzen */
    OR.w    #0040h,R0               ; /* re-enable interrupt after REIT */
    OR.b    (_uexc_current_func + 2),R0H
;   /* create stack frame */
    PUSH.w  R0                      ; /* FLG_h + PC_h */
    PUSH.w  (_uexc_current_func & 0ffffh)
    REIT                            ; /* return from interrupt */
    NOP
#pragma ENDASM
    }
```

Neben dem Stackframe wird noch die Adresse der Funktion *UEXC_KillSelf()* auf den Stack gelegt. Das ist notwendig, damit der Task, so er fertig ist, sich selbst aus der Taskliste entfernen kann. Wird ein laufender Task durch eine Interruptserviceroutine (ISR) unterbrochen, ist dies normalerweise unkritisch, da der Scheduler seinerseits ISRs nicht unterbrechen kann.

Im Falle des Monitors stellt sich dies anders dar. Der M16C kann über die serielle Schnittstelle UART1 mit einem PC gekoppelt werden. Auf dem PC läuft dann eine Debugsoftware, die das Programmieren des M16C mit der Applikation sowie eine schrittweise Programmabarbeitung gestattet. Das "Durchsteppen" durch die Applikation, man kann am PC-Monitor jede einzelne Programmzeile anspringen (Breakpunkte setzen) sowie Variablen ansehen oder modifizieren, nutzt spezielle Eigenschaften des Prozessors. Breakpunkte werden mit dem so genannten Address-Match-Register gesetzt. Eine Hardware im Schaltkreis vergleicht Programmzähler und Address-Match-Register und stoppt den Prozesser, sobald die Inhalte gleich sind.

Die Kommunikation zwischen PC und M16C erfolgt über die UART1. Daten oder Befehle über die UART1 werden vom Monitor ausgewertet. Der Monitor ist ein Stück Software, das vom Schaltkreishersteller vorgegeben wird. Empfangene Zeichen lösen den zugehörigen Interrupt aus. Damit die Zeichenbearbeitung, die ja einige Zeit in Anspruch nehmen kann, die Abfolge weiterer Interrupts nicht zu sehr beeinflußt, ist die UART1-ISR extrem kurz. Nach dem Zeichenempfang wird die ISR sofort beendet, aber anschließend sofort in den Monitor gesprungen und erst, wenn dort das Zeichen bearbeitet wurde, kehrt der Prozessor zu seiner eigentlichen Applikation zurück.

Der Monitor ist jedoch kein Task des UEXC-System, diesem ist das Vorhandensein des Monitors nicht einmal bekannt!

Wenn nun während dessen der Monitor aktiv ist, ein Task-Wechsel erfolgt, hat dies schwerwiegende Konsequenzen. Der Scheduler legt den aktuellen Inhalt des Stackpointers in den TCB des gerade deaktivierten Tasks und schaltet dann um. Dem neuen Task ist dies egal, wird aber der Task, in dessen TCB der Stackpointerinhalt des Monitor liegt, wieder aktiviert, geht die Programmkontrolle nicht mehr korrekt zurück an den Monitor. Die Funktion *UEXC_Resume()* setzt einen Interrupt-Stack-Frame voraus, der aber nicht gegeben ist - das System stürzt komplett ab. Es gibt nur eine Möglichkeit, dies zu verhindern: Im Monitor-Mode darf kein Scheduling stattfinden. Das Beispielprogramm demonstriert die Lösung:

```
/* Der Timer A3 kontrolliert den zyklischen Aufruf der Schedulers*/
void far vTimerA3ISR(void);
#pragma INTERRUPT vTimerA3ISR
void far vTimerA3ISR(void){
#pragma ASM
    ; /* check if we had interrupted the monitor */
        MOV.B       17[SP], R0H             ; /* get PC(H) */
        AND.B       R0H, 0FH                ; /* mask flags out of PC(H) */
        MOV.B       15[SP], R0L             ; /* get PC(M) */
        CMP.W       #0fc0H, R0              ; /* monitor interrupted ? */
        JLTU        ?+
        POPM        R0,R1,R2,R3,A0,A1,FB
        REIT                                ; /* ==>> do NOT switch task) */
        NOP
?:
        STC         SP, _uexc_current_sp    ; /* save stackpointer */
        JSR.a       _UEXC_CheckTask         ; /* Scheduler */
        POPM        R0,R1,R2,R3,A0,A1,FB
        REIT                                ; /* ==>> exit */
        NOP
#pragma ENDASM
    }
```

Damit steht nun ein außerordentlich kompaktes und stabiles Multitasking-RTOS zur Verfügung.

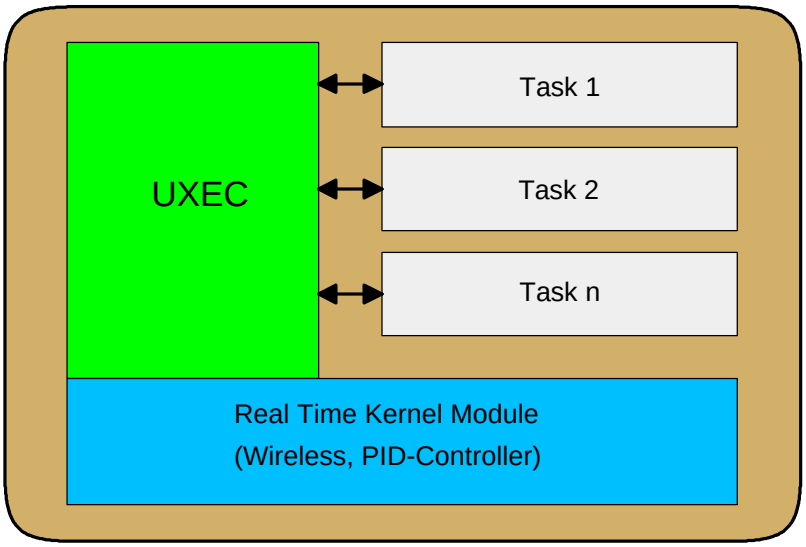



Bild 4.6 Aufbau des Roboterbetriebssystems

### 4.2.4 Regelalgorithmen für die Antriebsmotoren

Bevor die eigentlichen Regelalgorithmen gefunden und implementiert werden, ist es notwendig, sich über das Bewegungsmodell des Roboters klar zu werden. Die Anordnung der Antriebsmotoren als Differentialantrieb ermöglicht eine exzellente Beweglichkeit mit minimalem Manövrierraum. Ermöglicht wird dies u.a. auch durch die getrennte Steuerung beider Motoren. Von der exakten Regelung der Antriebe hängt die Bewegungspräszision, z.B. einer einfachen Geradeausfahrt, direkt ab.

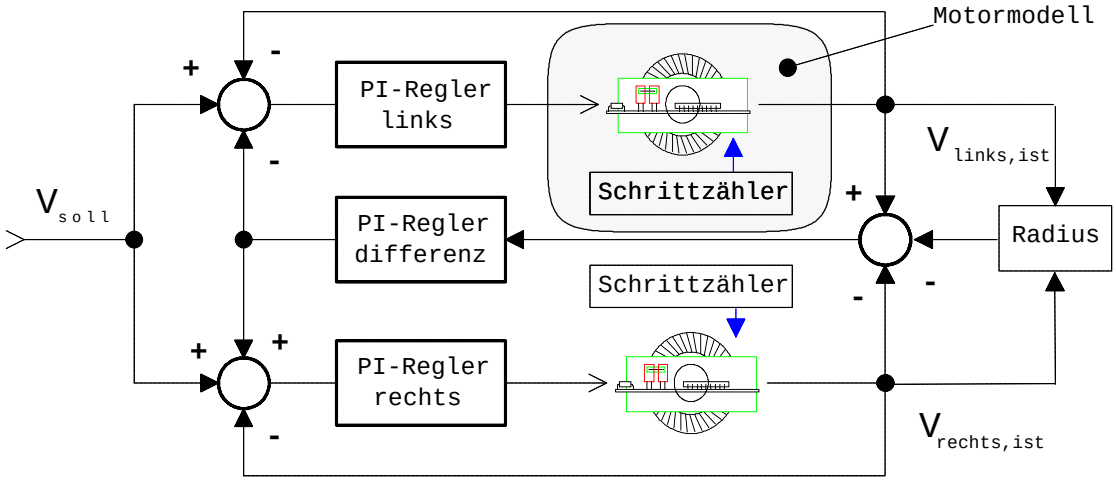



Bild 4.7 Blockschaltbild der Regler für die Antriebsmotoren

Zu den eigentlichen Motorreglern kommt noch ein weiterer Regler hinzu, der die Regelabweichung zwischen den Motoren korrigiert.
Der klassische Regler aus der Analogtechnik ist der PID-Regler. Dieser wird mit seinen drei Anteilen P (proportionaler Anteil, Verstärkung), D (differentieller Anteil, Änderungsgeschwindigkeit) und I (integraler Anteil, aufsummierter Fehler) beschrieben. Den mathematischen Zusammenhang verdeutlicht das folgende Integral:

$$x_a(t) = k_R \cdot \left[ x_d(t) + \frac{1}{T_N} \cdot \int_0^t x_d(\tau)\, d\tau + T_v \frac{dx_d}{dt} \right]. \qquad (4.1)$$

Für die Antriebsregelung im Roboter können jedoch keine kontinuierlichen Regler eingesetzt werden, der Regler muß zeitdiskret arbeiten. Die Überführung in einen zeitdiskreten PID-Regler wird in [22] gezeigt:

$$x_a(kT) = k_R \cdot \left[ x_d(kT) + \frac{T}{T_N} \cdot \sum_{v=0}^{k} x_d(vT) + \frac{T_v}{T} \left\{ x_d\,(kT) - x_d((k-1)\,T) \right\} \right] . \qquad (4.2)$$

In der Praxis erweißt es sich oft als schwierig die entsprechenden Parameter für P, I und D zu ermitteln. Empirische Versuche sind recht erfolgreich. Um die unumgänglichen Versuchsreihen abzukürzen und um Gefühl für die maximal erreichbaren Werte bzw. Optimierung zu bekommen, wird mit Hilfe von MathLAB-SIMULINK vor den praktischen Experimenten ein theoretisches Simulationsmodell des Regler entworfen. Im folgenden werden die unter [23] detailliert beschriebenen Ergebnisse kurz vorgestellt.

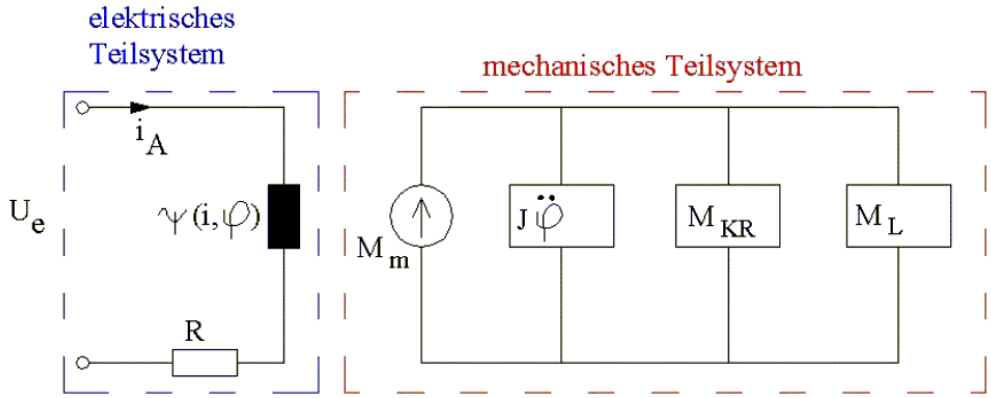



Bild 4.8 Komponenten des Motormodells

Basis der Simulation bildet das Motormodell eines DC-Motors. Es existieren zwei Teilsysteme, ein elektrisches und ein mechanisches. Das elektrische System lautet wie folgt:

$$u_e = i_A(t)R_A + L\frac{di_A}{dt} + c\Phi\frac{d\varphi}{dt} \ . \qquad (4.3)$$

Mit Hilfe des Drehimpulssatzes nach Newton/Euler definiert sich das mechanische System entsprechend:

$$M_m = J\frac{d^2\varphi}{dt^2} + \rho\frac{d\varphi}{dt} + k^*\varphi + M_L \ . \qquad (4.4)$$

Dabei ist $M_m$ das erzeugte Drehmoment, $M_L$ das Lastmoment, J das Trägkeitsmoment und p das Reibmoment. Der Zusammenhang von mechanischen und elektrischen Systemen läßt sich mit der Maschinengleichung, welche den lineare Zusammenhang von Ankerstrom und Drehmoment beinhaltet, so angegeben:

$$M_m = c\Phi i_A(t) \ . \tag{4.5}$$

Durch Einsetzung und Umformen mit anschließender Laplace-Transformation und Normierung ergibt sich:

$$G_{Motor}(p) = \frac{K}{p^2 T_m T_e + pT_m + 1} \ . \tag{4.6}$$

Die Konstante K ist die Verstärkung, $T_m$ die mechanische Zeitkonstante und $T_e$ die elektrische Zeitkonstante. Mit der Annahme, daß $T_m >> T_e$ reduziert sich das Motormodell zu:

$$G_{Motor}(p) = \frac{K}{(1 + pT_m)(1 + pT_e)} \ mit \ K = \frac{1}{c\Phi}, \ T_m = \frac{RJ}{(c\Phi)^2}, \ T_e = \frac{L}{R} \ . \tag{4.7}$$

Damit dieses Modell in MathLAB/SIMULINK getestet werden kann, muß es zusammen mit der Übertragungsfunktion des Abtastgliedes in den Z-Bereich transformiert werden. Die allgemeine Übertragungsfunktion der Regelstrecke mit dem Einsatz realer Werte für die Konstanten lautet dann:

$$G_S(z) = \frac{a_1 z + a_0}{b_2 z^2 + b_1 z + b_0} = 13,38 \frac{z+0,488}{(z-0,8187)(z-0,1353)} \ . \tag{4.8}$$

Als Regler wird ein PI-Regler eingesetzt. Ein PID-Regler wäre theoretisch besser, aber im Vorgriff auf die reale Implementation, wird hier schon auf die Limitierung der vorhandenen Rechenleistung verwiesen; die Berechnung des D-Anteils ist mit Divisionen verbunden, welche auf *Embedded Controllern* nach Möglichkeit vermieden werden. Außerdem verringerte der D-Anteil in dem System der drei gekoppelten Regler die Stabilität. Die Schwingneigung der drei Regler nahm zu, so daß auf die geringe Verminderung der Einschwingzeit eines vollständigen PID-Reglers gegenüber einem PI-regler verzichtet wurde.
Die Übertragungsfunktion des PI-Reglers ist:

$$G_R(z) = k_R \frac{z-a}{z-1} \text{ mit } a = (1 - \frac{T}{T_N}) \ . \tag{4.9}$$

Mit diesen Werten kann nun die Sprungantwort in MathLAB/SIMULINK berechnet werden. Die Konstante a = 0,8187 ist so gewählt, daß sie den größten Pol des Motors kompensiert, $k_R$ wird interativ optimiert. Die Abtastzeit T = 50 ms ist vom System her vorgegeben, $T_N$ ergibt sich deswegen als $T_N$ = 276 ms.

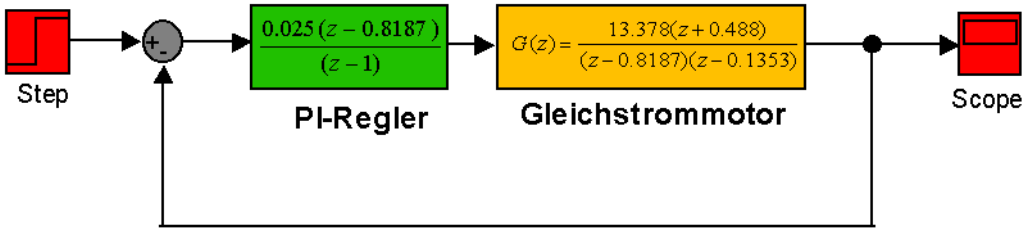



Bild 4.9 Blockschaltbild für MathLAB/SIMULINK

Die Abbildung 4.10 zeigt die Sprungantworten der drei untersuchten Regler (P - untere Kurve, blau; PI - obere Kurve, grün; PID - mittlere Kurve, rot).
Das leichte Überschwingen des PI-Reglers wird im Interesse der besseren Implementation hingenommen. Anhand der Simulationsergebnisse können nun die real erreichten Werte eingeschätzt werden. Der nächste Schritt ist deshalb die Umsetzung der Formeln in ein passendes Programm. Hier steht die klare Forderung nach reiner Integer-Mathematik, d.h. alle Berechnungen müssen mit ganzen Zahlen im Wertebereich von max. *long integer* erfolgen.
Als Ansatz zur Implementierung dient die Berechnungsvorschrift nach (4.2). Da der differentielle Anteil entfällt, vereinfacht sich die Gleichung zu:

$$x_a(kT) = k_R \cdot \left[ x_d(kT) + \frac{T}{T_N} \cdot \sum_{v=0}^{k} x_d(vT) \right]. \qquad (4.10)$$

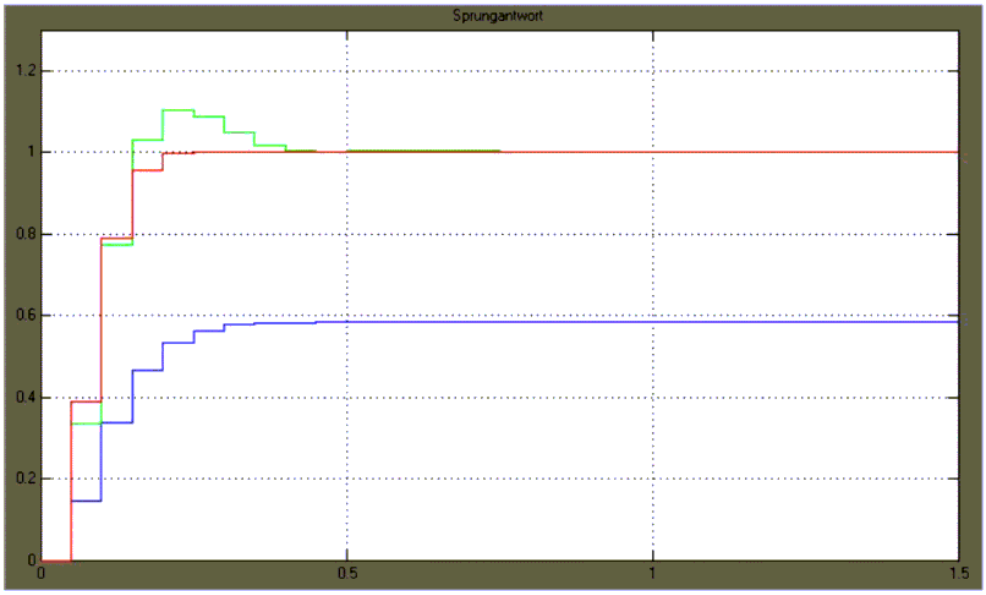



Bild 4.10 Sprungantworten von P, PI und PID-Regler

Die Konstanten $k_R$, T und $T_N$ sind zwar formal die gleichen wie die in MathLAB/SIMULINK ermittelten, nützen in der realen Anwendung jedoch nichts, da hier normierte Zahlen benötigt werden. Bei einem PI-Regler werden die Verstärkung und der aufsummierte Fehler zur Berechnung der Regelgröße verwendet. Die Gleichung wird in eine dem Rechner verständliche Form überführt:

$$wPwm = nKr \cdot (wSoll - wIst) + iDelta + nKi \cdot SUM(wSoll - wIst)\ . \qquad (4.11)$$

In dieser Gleichung sind *nKr* und *nKi* ganzahlige Konstanten, *iDelta* ist die Differenzgeschwindigkeit der Motoren sowie *wSoll* und *wIst* die Vorgabewerte der Regelung. Das Beispiel zeigt die lauffähige Implementierung.

```
/*********************************************************************
* Kurzbeschreibung     : Main-Loop, alle 50 ms aktiv
*                        der PID-Regler arbeitet nach folgendem Verfahren:
*
*  wPwm = Kr * (Soll - Ist) + Ki * SUM[Soll - Ist]
*
* Laufzeit             : ca. 35 µs bei 10 MHz - Takt
* Autor                : Jens Altenburg
*********************************************************************/
void DCM_vMain(void)
    {
    ...
    /* Differenzregler */
    iDelta = wSpeedRe;
    iDelta -= wSpeedLi;/* Differenzgeschwindigkeiten der Motoren */
    iDelta >>= 2;
    iDelta = iDelta - iDiffReLi;                  /* P-Anteil Differenz   */
    iIntegralDiff += iDelta;                      /* I-Anteil aufsummieren */
    if(iIntegralDiff >  1023) iIntegralDiff =  1023;
    if(iIntegralDiff < -1023) iIntegralDiff = -1023;
    iDelta += iIntegralDiff;


    /* PI-Regler rechter Motor */
    i32  = wSpeedRe;
    i32 -= wSpeedDebitRe;                          /* P-Anteil in i32 */
    i32 += iDelta;                                 /* Differenzregler */
    i32IntegralRe += (i32 >> 4);                   /* nKr = 0.0635    */
    if(i32IntegralRe > 0x0000fff0) i32IntegralRe = 0x0000fff0;/*Limitierung*/
    if(i32IntegralRe < 0x0000000f) i32IntegralRe = 0x0000000f;
    i32 >>= 2;                                     /* nKi = 0.25      */
    i32 += i32IntegralRe;                          /* Aufsummierung   */
    TA2 = (word)i32;                               /* Ausgabe         */

    /* PI-Regler linker Motor */
    ...
    }
```

Die Konstanten nKr und nKi können nur iterativ gefunden werden. Da bei falscher Dimensionierung der Regler zum Schwingen neigt, wird mit der Erprobung von nKr (P - Anteil) begonnen (nKi = 0). Erreicht der Regler seinen stationären Zustand, wird nKi vergrößert und nKr verringert. Damit bei Zahlenwerten kleiner 0 nicht dividiert werden

muß, kann entweder die Abtastzeit geändert werden oder, falls das nicht möglich ist, ersetzen Schiebeoperationen die Division. Hier ist bei der Portierung des Codes auf andere Compiler streng darauf zu achten, daß Schiebeoperationen von *signed integer* Werten vorzeichenrichtig erfolgen (das wird vom C Standard nicht definiert).
Der implementierte Regler arbeitet zuverlässig sowie genau genug bei einem Rechenzeitbedarf von nur 35 µs in einem 50 ms Raster. Die geringe Rechenzeit ist wichtig, da der Regler als Echtzeittask im Interrupt läuft und den Zeit-Jitter klein hält.

### 4.2.5 Funkkommunikation

Das hardwareseitige Interface des Funkmodems ist im Abschnitt 3.3.5 erläutert. Nun geht es darum, die softwaretechnischen Randbedingungen für eine sichere Funkübertragung zu schaffen.
Die drahtlose Funkdatenübertragung ist eine bequeme Möglichkeit, Kommunikationsverbindungen zu autonomen Systemen zu unterhalten. Es sind keine hinderlichen Kabelverbindungen notwendig und im Gegensatz zur drahtlosen optischen Datenübertragung muß kein permanenter Sichtkontakt bestehen. Nachteilig ist jedoch die relativ große Fehlerempfindlichkeit. Zündende Neoröhren, Funkenabriß bei Schaltkontakten bzw. das Bürstenfeuer von DC-Motoren stellen erhebliche Störpotentiale bereit. Speziell letzteres, das Bürstenfeuer schlecht entstörter Motoren, äußert sich in sehr breitbandigen, energiereichen Störemissionen. Finden diese als Antriebsquellen für den Roboter Verwendung, muß durch sorgfältige Entstörung das Erzeugen von Bürstenfeuer unterbunden werden. Hierbei ist darauf zu achten, daß die Dämpfung direkt am Motor erfolgt, jeder Zentimeter Anschlußleitung wirkt sonst als unbeabsichtigte Antenne.
Neben der hardwareseitigen Störunterdrückung kann ein fehlertolerantes Funkprotokoll für bessere Übertragungssicherheit sorgen. Fehlertoleranz führt im allgemeinen jedoch zu einem gewissen "Datenoverhead", der speziell in der vorliegenden Anwendung nicht gewünscht ist.
Die Roboter bewegen sich innerhalb einer Modelllandschaft die eine maximale Ausdehnung von ca. 10 m x 10 m besitzt. Unter diesen Umständen ist in jedem Fall eine ausreichende Feldstärke auf der gesamten Spielfläche gegeben. Empfangsversuche haben gezeigt, daß unter diesen Bedingungen die Feldstärken der anderen Störquellen, wie Schaltimpulse, Neonröhren oder Personalcomputer nur eine untergeordnete Rolle spielen. Beträchtlich stärker macht sich die Taktfrequenz des Oszillators des Steuercontrollers bemerkbar.

Die Frequenz von 16 MHz erzeugt offenbar kräftige Oberwellen. Bei der Softwareerprobung im Emulator (8-Bit-Controller, MauSI 1) war offensichtlich die 27. harmonische Oberwelle Ursache eines kompletten Funkausfalls.
Abhilfe brachte letztlich nur die Ankopplung einer räumlich abgesetzten Antenne über eine Linkleitung zum Funkmodul.
Aus diesen Erfahrungen heraus wurde besonderer Wert auf das Oszillatordesign des 16-Bit-Controllers gelegt. Die Anordnung des Quarzes erfolgte so dicht wie möglich am Schaltkreis. Der Quarz und alle Taktleitungen wurden zudem mit einer Massefläche abgeschirmt und gleichzeitig der Abstand Controller - Funkmodul maximiert.
Die Schilderung dieser Störungen im Zusammenhang mit dem Softwaredesign hat folgenden Hintergrund: Ursprünglich sollte die Funkdatenübertragung Event-getrieben sein. Die Auswertung von Datensignalen des Modules sollte durch das Auftreten von *carrier detect* Signalen aktiviert werden. Der Anteil der fälschlich erkannten *carrier detect* Impulse war so hoch, daß ohne eine weitere Dateninterpretation dieses Signal als alleiniger Indikator unbrauchbar war.
Da alle Funkmodule auf der gleichen Frequenz arbeiten und trotzdem sichergestellt werden muß, daß jeder Sender jeden Empfänger erreichen kann, wurde folgendes Funkprotokoll definiert. Anhand einer Grafik läßt sich das Procedere einfach erläutern.

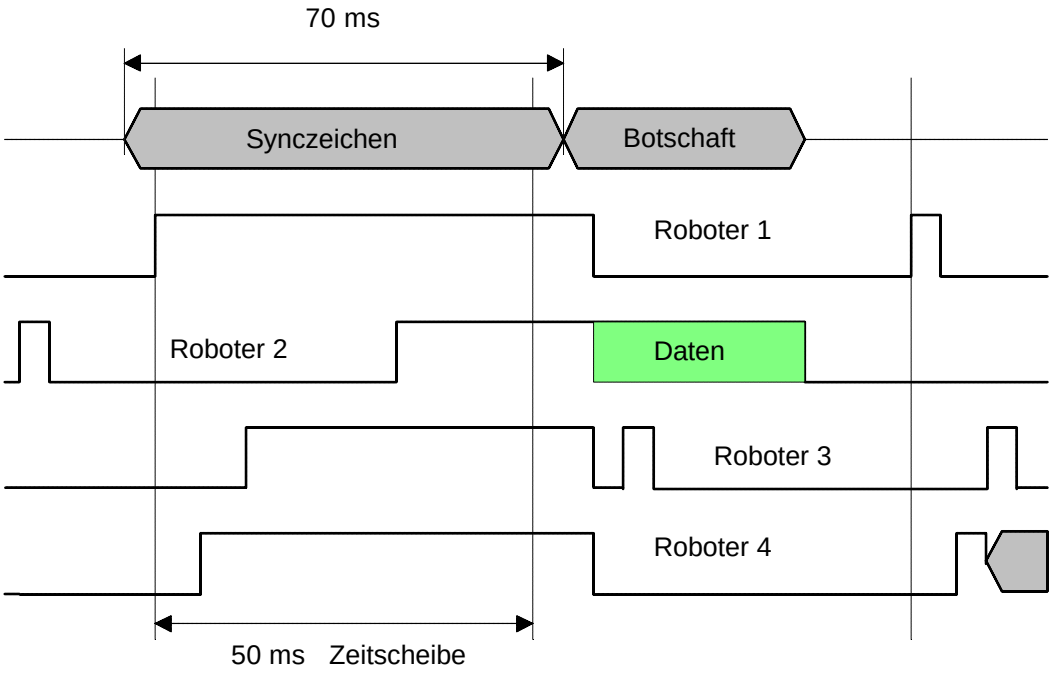
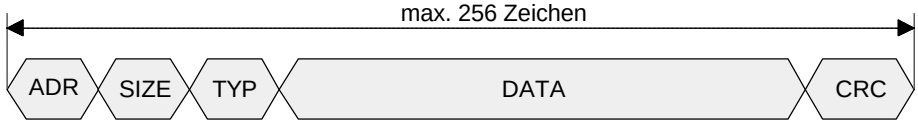



Bild 4.11 Zeitdiagramm des Funkprotokoll

Auf der Basis eines Zeittaktes von 50 ms werden alle Empfänger zyklisch kurz eingeschalten. Diese Zeitscheiben sind zwischen den Roboter nicht synchronisiert. Das Synczeichen, bestehend aus der Zeichenfolge 0x55, 0xff, 0x55, 0xff,... überlappt die Zeitscheiben (ca. 70 ms lang). Damit wird sichergestellt, daß alle empfangsbereiten Roboter innerhalb ihrer Zeitscheibe den eigenen Empfänger aktivieren.
Innerhalb der Botschaft wird der eigentliche Empfänger spezifiziert, d.h. im Beispiel bleibt dann nur noch Roboter 2 aktiv, und empfängt das Datenpaket. Da Sender und Empfänger auf der gleichen Frequenz arbeiten, verhindert ein Sperrmechanismus, daß zwei Sender gleichzeitig aktiv sind. Bevor eine Sendung abgestrahlt wird, testet der sendebereite Roboter das Vorhandensein eines weiteren Trägersignals.
Erst wenn kein weiterer Träger detektiert wird, kann die eigene Botschaft abgestrahlt werden. Nur dann, wenn zwei Roboter zum exakt gleichen Zeitpunkt einen Sendeversuch unternähmen, würde dieses Verfahren scheitern. Wegen der kleinen Zeitspanne zwischen Test des Trägersignales und Start einer Übertragung, ist Wahrscheinlichkeit gleichzeitiger Sendeversuche gering (keine Störungen in der Praxis beobachtet).
Die übertragene Botschaft unterteilt sich in verschiedene Abschnitte. Neben der bereits erwähnten Empfängeradresse werden die Länge der Botschaft, der Typ, das Datenfeld und eine CRC-Summe übermittelt.




| | |
|---|---|
| ADR | Roboteradresse |
| SIZE | Zeichenanzahl (max. 256) |
| TYP | Option (normalerweise 0x00) |
| DATA | Dateninhalt |
| CRC | Prüfsumme |

Bild 4.12 Aufbau einer Funkbotschaft

Zum Zugriff auf empfangene Daten bzw. zum Absenden von Botschaften gibt es zwei Funktionen, *WRC_vRxMessage()* und *WRC_vTxMessage()*.
Mit der Abfrage *WRC_stCheckModem()* wird geprüft, ob Daten empfangen wurden. Es ist prinzipiell auch möglich, über eine Botschaft einen neuen Task zu starten. Mit Hilfe des *TYP-Bytes* aktiviert das Funkmodem einen neuen Task. Das *TYP-Byte* ist dabei ein Index auf einen Task aus einer fest im ROM des Roboters abgelegten Taskliste. Mit diesem Mechnismus kann beispielsweise per Funkbefehl die komplette Kontrolle über den Roboter übernommen werden.

### 4.2.6 Einfache Bildverarbeitung und Mustererkennung

Ein wesentliches Leistungsmerkmal des Robotersystems "MauSI 2" ist die Kombination des Roboters mit einer CMOS-Kamera. Dieser Sensor ergänzt die Sensoraustattung um einen besonders leistungsfähigen optischen Sensor.
Ursprünglich sollte die Kamera lediglich Bilddaten aufnehmen und sie per Funkmodem zur weiteren Verarbeitung an einen PC senden. Wegen der zu geringen Bandbreite des Funkmodems ist keine Streaming-Videoübertragung möglich. Es müssen Einzelbilder temporär im RAM des Controllers zwischengespeichert werden, die dann entsprechend der vorhandenen Funkkapazität an den PC übermittelt werden. Infolge der begrenzten RAM-Größe werden nur niedrig aufgelöste Bilder übertragen.
Unter der Voraussetzung, daß die Bilder keine bewegten Objekte darstellen, können auch höher aufgelöste Bilddaten (356 x 256 Pixel) gesendet werden. In diesem Fall wird ein Gesamtbild in Segmente aufgeteilt, die dann sequentiell übertragen werden. Ein Bild in der Auflösung von 356 x 256 Pixel ergibt 8 Segmente von je 356 x 32 Pixel. Die Datenübertragung eines Segmentes benötigt etwa eine Sekunde, ein Gesamtbild wird demzufolge in ca. 8 Sekunden übertragen. Das Funkmodem arbeitet dabei mit einer Datenrate von 115200 Baud in einem Spezialmode. In diesem Mode wird nur ein Minimum an zusätzlichen Sync-Daten übermittelt.
Durch eine Veröffentlichung in [23] aufmerksam geworden, entstanden eigene Versuche zur Bildverarbeitung direkt auf dem Roboter. Ziel der Versuche war die Nutzung des Kamerasensors zur einfachen Musterdetektion und die Extraktion von Korrekturdaten zur Steuerung des Roboters, so daß dieser einem einmal erkannten Objekt selbstständig zu folgen vermochte.

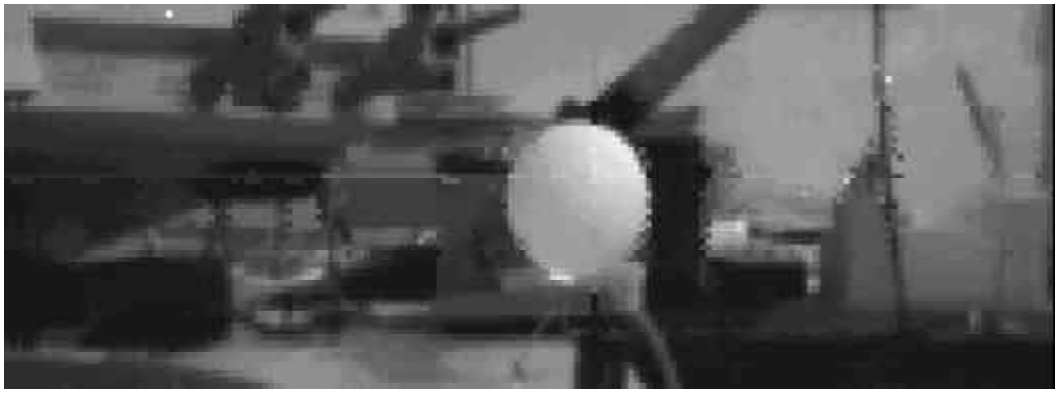
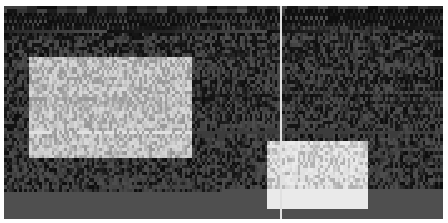


Bild 4.13 Kamerabild des Roboters mit Tennisball als Zielobjekt (176 x 64 Pixel)

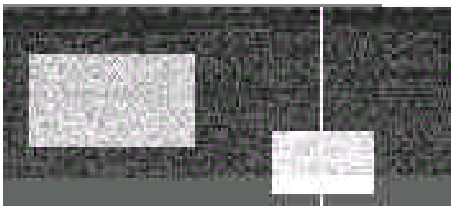
Die Komplexität des zu erkennenden Objektes bestimmt die Komplexität des Filteralgorithmus zur Objektdetektion. Ein denkbares Verfahren besteht in der Kantendetektion. Aus der Lage und Anordnung aller Kanten wird dann auf das Objekt und auf seine Lage geschlossen [24].
Ohne tiefere Erläuterungen wird deutlich, daß ein solches Vorgehen enorme Ressourcen hinsichtlich Kantendetektion und nachfolgender Datenanalyse erfordert.
Es ist also wichtig, im Vorfeld Aussehen bzw. Eigenschaften der zu erkennenden Objekte zu definieren. Im Zusammenhang mit der Markierung und Beobachtung von Robotern mittels einer über dem Spielfeld angeordneten Kamera [20] sind gute Erfahrungen mit der Erkennung und Lagemarkierung von Kreisen gemacht worden.
Der Kreis als relativ einfaches Objekt ist dadurch gekennzeichnet, daß sich alle zum Kreis gehörenden Bildpunkte um den Kreismittelpunkt herum anordnen. Die Kreiserkennung ist zudem rotationsinvariant.
Die farbigen Kreismarkierungen zur Lokalisation der Roboter in [20] werden durch farbige Tischtennisbälle, die auf dem Roboter plaziert werden, ersetzt. Die Kugel wird aus jedem Blickwinkel von der Kameraoptik als Kreisfläche auf dem CMOS-Sensor abgebildet.
Im Ergebnis der Bildverarbeitung steht die Aufgabe, die Abbildung der Kugel (Kreisfläche) auf der Sensorfläche zu lokalisieren und von anderen Bildsegmenten zu trennen. Der Schwerpunkt der segmentierten Kreisfläche ist als X-Koordinate ein Maß für die Winkelabweichung bei der Zielverfolgung. Da die Größe der Kugel fest definiert und bekannt ist, kann über eine Inhaltsberechnung der segmentierten Kreisfläche eine Entfernungschätzung erfolgen.
Für die Bildsegmentierung ist es erforderlich, das diskrete Bildsignal als Menge aller Punkte $M_{Pixel}(m, n)$ in zusammenhängende Teilmengen $T_{Bild}^{k}(i^k, j^k)$ zu gliedern. Im konkreten Einsatzfall beträgt m = 176, n = 64. Die Anzahl *k* der Teilmengen $T_{Bild}^{k}$ soll *k=1* erreichen (Zielobjekt gefunden) und es ist notwendig, den Schwerpunkt von $i^k$ zu definieren, um die Teilmenge im Bild zu lokalisieren. Zur Segmentierung wird ein Schwellwertoperator benötigt [25]:

$$g(m, n) = \begin{cases} I_1 \text{ für } 0 \le f(m, n) < s \\ I_2 \text{ für } s \le f(m, n) \le f_{max} \end{cases} . \qquad (4.12)$$

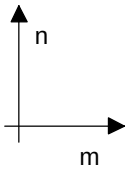
$I_1$, $I_2$ sind zwei beliebige, jedoch voneinander verschiedene Werte, s ist die Intensitätsschranke. Mit diesem einfachen Schwellwertoperator wird nun ein Test unternommen.

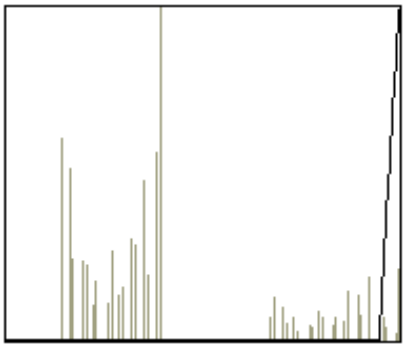


s = p + ((s - p) >> 1);



s = 0x25;



Histogramm des Testbildes

Schwerpunktermittlung bei unterschiedlichen Schwellwertoperatoren
oben - s über gleitenden Mittelwert berechnet
unter - s nach Versuch fest eingestellt

Bild 4.14 Schwerpunktermittlung mit unterschiedlichen Schwellwerten

Anhand eines Testbildes wird untersucht, welche Auswirkungen der Schwellwertoperator auf die Genauigkeit der Bildsegmentierung hat. Im Testbild (siehe Abbildung 4.14) sind zwei Rechecke als potentielle Ziele zu erkennen. Das "richtige" Ziel ist jenes mit der größeren Intensität (Rechteck rechts unten im Bild).
Um Rechenzeit zu sparen, wird über einen kleinen Teil des Bildes (Bildmitte) der Mittelwert gebildet und aus diesem und dem Spitzenwert dieses Bereiches eine Schwelle definiert. Mit diesem Wert wird über alle Bildzeilen der Intensitätsschwerpunkt gebildet. Der Mittelwert der 10 (diese Zahl ist empirisch definiert) größten Schwerpunkte ist dann der Schwerpunkt des zu detektierenden Objektes (es interessiert nur der Wert der m-Koordinate).
Im oberen Bild ist ein deutlicher Fehler bei der Lokalisierung feststellbar. Zur Kontrolle des Algorithmus zur Schwerpunktbildung wird das Histogramm des Testbildes ausgewertet und daraus der optimale Schwellwert in die Berechnung einbezogen.
Im unteren Bild wurde das Objekt dann korrekt lokalisiert. Mit der richtigen Berechnung des Schwellwertes ist die Genauigkeit der Objektlokalisation eng verknüpft.
Einen Ansatz zur Ermittlung eines optimalen Schwellwertes wird in [25] geliefert. Es sollen folgende Voraussetzung angenommen werden: Es existiert ein Bildfeld mit der Ausdehnung *M x N (M = 120, N = 64)*. Das gesuchte Objekt ***O*** hebt sich durch Intensitätsunterschiede vom Hintergrund ***H*** ab (helles Objekt auf dunklem Grund). Die

Auftrittswahrscheinlichkeit der Intensitätswerte ***i*** ist in beiden Bildbereichen normalverteilt. Für die Verteilungsfunktion gilt somit:

$$P_\xi(i) = \frac{1}{\sqrt{2\pi\sigma_\xi}} \exp(-\frac{(i-\mu_\xi)^2}{2\sigma_\xi^2}) \; . \qquad (4.13)$$

$\xi \in \{0, H\}$, $\mu_\xi$ *Mittelwert*, $\sigma_\xi$ *Steuung*

Weiterhin soll gelten: $p_O$ ist die Anzahl der Bildpunkte in $f_O$ sowie $p_H$ die Anzahl der Bildpunkte in $f_H$. Damit gilt:

$$\frac{p_O+p_H}{M \cdot N} = 1 \; . \qquad (4.14)$$

Daraus ergibt sich die Intensitätsverteilung des gesamten Bildes zu:

$$p(i) = \frac{p_O \cdot P_O(i)+p_H \cdot P_H(i)}{M \cdot N} \; . \qquad (4.15)$$

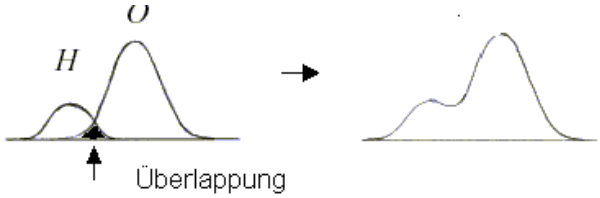



Bild 4.15 Grafische Darstellung der Gleichung 4.15

Da sich die Bereiche von ***H*** und ***O*** überlappen, ist mit einem Segmentierungsfehler zu rechnen. Der Gesamtfehler ermittelt sich zu:

$$E(S) = (p_O \cdot E_O(S) + p_H \cdot E_H(S)) / (M \cdot N) \quad mit\ S - Schwellwert \; . \qquad (4.16)$$

Aus den Gleichungen 4.13 und 4.16 läßt sich nach Umstellen, Ableiten nach S und Fehlerminimierung E = 0, der optimale Schwellwert wie folgt errechnen:

$$S = \frac{\mu_O+\mu_H}{2} - \frac{\sigma^2}{\mu_O-\mu_H} \ln(\frac{P_O}{P_H}) \; . \qquad (4.17)$$

Diese Gleichung bildet die Basis zur Ermittlung des Schwellwertes zur Trennung von Objekt und Hintergrund. In der vorliegenden Form ist sie jedoch als Berechnungsvorschrift noch nicht brauchbar.
Zu einigen Werten lassen sich Abschätzungen treffen. Der Intensitätsmittelwert $\mu_O$ ist immer größer als $\mu_H$, da das Objekt heller als der Hintergrund ist. Die Größe des zweiten

Terms der Gleichung 4.17 wird maßgeblich von der Streuung der Intensitäten bestimmt. In [25] wird davon ausgegangen, daß die Streuung von Objekten gleich der des Hintergrundes ist. Damit der zweite Term so klein wird, daß er vernachlässigt werden kann, wird eine zusätzliche Vereinfachung getroffen. Das zu suchende Objekt wird als flächiger Lambertscher Strahler (Objekt mit gleichmäßiger Helligkeit) aufgefaßt, damit wird die Intensitätsstreuung $o^2 << (\mu_O - \mu_H)$ und somit entfällt der zweite Teil der Gleichung. Das folgende Beispielprogramm zeigt eine mögliche Implementierung der Ermittlung des Schwellwertes zur Bildsegmentierung.

```
/*************************************************************************
* Kurzbeschreibung     : Suchfunktion nach größtem Object
* Uebergabeparameter   : Farbparameter
* Return Value         : x-Koordinate
* Autor                : Jens Altenburg
*************************************************************************/
VIDEO_stObject VIDEO_wSearchObject( byte bFarbe, enPictureTyp enTyp )
    {
    ...
    /* Intensitätsmittelwert des Bildes bilden, zur Rechenzeitminimierung
       wird nur jede vierte Bildzeile in Rechnung einbezogen
       gleichzeitig wird der Spitzenwert der Helligkeit gesucht */
    if(enTyp = enMonoPicture)
        {
        bLevel = 0;                       /* Spitzenwert μ₀ */
        k = 0;                            /* Mittelwert über Bild μH */
        for(n = 0; n < nMonoResolution; n+= 4)
            {
            wXpos = 0;
            for(m = 0; m < nPixelResolution; m++)
                {
                wXpos += bMonoPix(m, n);  /* Werte aus Zeile aufsummieren */
                if(bMonoPix(m, n) > bLevel)         /* Spitzenwert finden */
                    {
                    bLevel = bMonoPix(m, n);
                    }
                }
            }
        k += ((byte)(wXpos / nPixelNumber);            /* Bildmittelwert */

        /* Abbruch der Suchfunktion bei zu kontrastarmen Bildern */
        if((bLevel - k) < 20) return eError;     /* zu schwacher Kontrast */

        /* Schwellwert aus Gleichung 4.17 festlegen */
        bLevel = k + ((bLevel - k) >> 1);        /* Schwellwert festlegen */
```

Die Gleichung 4.17 wird in die Berechnungsvorschrift *bLevel = k + ((bLevel - k)>> 1);* umgesetzt.

Die CMOS-Kamera ist mit eigener Intelligenz hinsichtlich der optimalen Verstärkungseinstellung der integrierten Sensormatrix ausgestattet (Belichtungseinstellung). Diese Steuerung versucht den Dynamikbereich der Sensoren an die Beleuchtungssituation anzupassen. Gelingt dies unter ungünstigen Umständen nur eingeschränkt, d.h. der Kontrast wird sehr klein, bricht die Suchroutine mit einer Fehlermeldung ab.

Detektion eines Tischtennisballes als lambertscher Strahler in ca. 25 cm Entfernung



Der gleiche Ball in einer Entfernung von ca. 60 cm

Der senkrechte weiße Strich zeigt die x-Koordinate des detektierten Objektes

Bild 4.16 Beispiele für Objektsegmentierung und Erkennung mit der CMOS-Kamera

Anhand einiger Versuche wurde die Erkennungssicherheit der Software getestet. Unter Modellbedingungen konnte eine gute bis sehr gute Erkennungssicherheit erreicht werden. Über Farbfilter werden verschieden eingefärbte Tischtennisbälle getrennt erkannt. Problematisch ist die starke Empfindlichkeit der Kamera im Infrarotbereich. Starke Infrarotstrahler, wie z.B. Glühlampen, führen zu Fehlinterpretationen. Hier schafft der Einsatz von IR-Sperrfiltern deutliche Verbesserungen.

### 4.2.7 Weiterführende Experimente mit Farbbildern

Trotz der relativ guten Performance des eingesetzten Mikrocontrollers sind der exakten Objektdetektion innerhalb von Monochrombildern Grenzen gesetzt. Damit die Informationen aus den Bilddaten für die Navigation genutzt werden können, muß die Bildverarbeitung in Echtzeit erfolgen.
Die Analyse von Monochrombildern zeigt hier ihre Grenzen. Bessere Ergebnisse sind erst bei der Auswertung höher aufgelöster farbiger Bilder zu erwarten.
Der eingesetzte Sensor hat eine maximale Auflösung von 352 x 288 Pixel. Ein komplettes Bild mit dieser Auflösung benötigt ein Speichervolumen von 202752 Byte. Eine solche Datenmenge kann im Controller nicht gepuffert werden. Zudem würde die unkomprimierte Übertragung über das serielle Interface bei 115200 Baud eine Sendezeit von ca. 18 Sekunden erfordern. In der Praxis sind diese Speichergrößen und Übertragungszeiten durch den Roboter (noch) nicht handhabbar.

Dennoch wurden einige Versuche mit dem Kamerasensor im QCIF Mode unternommen. In diesem Fall wird ein Bild mit einer Auflösung von 176 x 144 Pixel erzeugt. Der Speicherbedarf beträgt dann nur noch 25344 Byte (25 %). Die Übertragungszeit verringert sich dementsprechend.
Die Abbildung 4.17 zeigt ein Farbbild der CMOS-Kamera mit der Auflösung von 176 x 144 Pixel.

Bild 4.17 Farbbild des Kamerasensors mit 176 x 144 Pixel

Trotz der Datenreduktion ist die Datenübertragung immer noch ein relativer Schwachpunkt. Die Bilddaten werden mit einer wesentlich höheren Geschwindigkeit vom Sensor geliefert, als die serielle Schnittstelle diese weiter geben kann. Selbst in der geringsten Abtastrate fallen fünf mal mehr Daten an, als zeitgleich gesendet werden können.
Um diesem Dilemma zu entgehen, wird ein Bild spaltenweise übertragen, d.h. vom ersten Bild werden die Spalten 0...34, vom zweiten Bild die Spalten 35...69, usw. übertragen. Bei Standbildern funktioniert dies ausgezeichnet (siehe Abbildung 4.17).

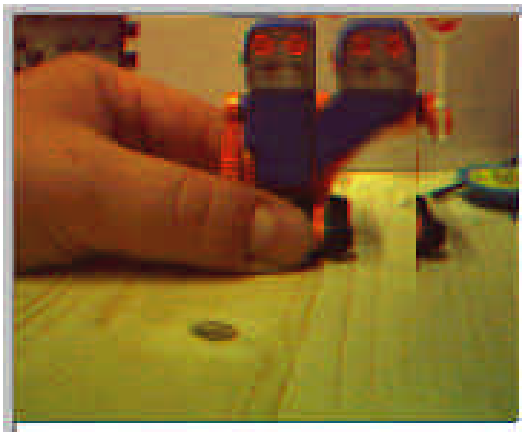


Bild 4.18 Bildartefakte bei bewegten Objekten

Wird jedoch der Spielzeugroboter bewegt, entstehen erheblich Bildstörungen. Die Abbildung 4.18 zeigt diese. In der Abbildung ist auch der spaltenweise Bildaufbau gut zu erkennen.
Der Roboter im Bild wird mit der Hand von rechts nach links gezogen. Der Bildaufbau erfolgt jedoch von links nach rechts.
Die Spalten 0, 1 und 2 sind völlig ungestört, die Hand die den Roboter bewegt ist einwandfrei zu erkennen. In der dritten Spalte ist der Kopf der Roboter noch in Ruhe, der Rumpf jedoch bereits in Bewegung. Erkennbar ist dies durch die Krümmung, die infolge der endlichen Abtastzeit von Bildzeile zu Bildzeile entsteht.
Die Spalte 4 zeigt den Roboter noch völlig unbewegt, das Fahrgestell ist noch stückweise zu sehen.
Die bei dieser extrem langsamen Bildabtastung von ca. einem Bild pro Sekunde entstehenden Fehler, gekoppelt mit den Artefakten aus der Bildsegmentübertragung aufeinanderfolgender Einzelbilder läßt die Schwierigkeiten der Bildverarbeitung erahnen. Um diese Probleme zu minimieren, liegt der Schwerpunkt weiterführender Untersuchungen auf der Definition effektiver Datenkompressionsalgorithmen und einer Leistungssteigerung bei der Datenübertragung.
Bei der Datenübertragung zeichnen sich Lösungen durch den Wechsel des Frequenzbandes von 433 MHz zum 2,4 GHz-ISM-Band ab. In diesem Bereich werden integrierte Transceiver-Bausteine mit Datenraten bis zu 1 MBaud verfügbar.

### 4.2.8 Lagebestimmung und Kursrechner

Die gerichtete sinnvolle Navigation im Raum ist ein wesentliches Kennezeichen intelligenter autonomer Systeme. Eingangs wurden verschiedene Arten der Navigation erläutert. Es wurde u.a. auch gezeigt, daß unter bestimmten Umständen selbst scheinbar chaotische Bewegungsmuster zu sinnvollen Reaktionen führen (Elsi).
Die exakte Selbstlokalisation autonomer Systeme ist an das Vorhandensein bestimmter Sensoren geknüpft. Je nach Umgebung und geforderter Genauigkeit kann der Roboter seine Position durch Einpeilen bestimmter "Landmarken" (charakteristische Punkte der Modellumgebung) oder durch eine Absolutmessung mit Hilfe eines GPS-Empfängers ermitteln.
Für das Robotersystem "MauSI" sind diese Methoden ungeeignet. GPS funktioniert innerhalb geschlossener Räume nicht und wäre zudem zu ungenau, die Erkennung charakteristischer Punkte und eine daraus hervorgehende Positionsberechnung ist

ebenfalls nicht vorgesehen. Der Roboter ist auf die Fremdbestimmung seiner Anfangsposition angewiesen.
Die internen Sensoren des Roboters, Radencoder und Beschleunigungssensor, ermöglichen jedoch eine fortlaufende Koppelnavigation. Theoretisch wäre auf der Basis erkundeter hindernisfreier Strecken und deren Abgleich mit hinterlegten Kartendaten eine Selbstlokalisation denkbar. Dieser Ansatz ist jedoch nicht weiter verfolgt worden, es besteht die Vermutung, daß zwar Kartendaten im ROM des Controllers abgelegt werden könnten, der Mustervergleich mit spezifischen Streckenabschnitten jedoch die Rechenkapazität des Controller übersteigt.
Der Roboter bewegt sich im zweidimensionalen Raum. Dabei bewegt er sich entweder rein translatorisch (Geradeausfahrt), rein rotatorisch (Drehung) oder in einer Kombination beider Elemente (Kreisbogen). In jedem Fall müssen die Positionsänderungen erfaßt und für weitere Berechnungen zur Verfügung gestellt werden.
Zur Messung des zurückgelegten Weges stehen die Radencoder und der Beschleunigungssensor zur Verfügung. Weiterhin benötigt der Kursrechner Informationen zur eingestellten Drehrichtung der Motoren. Auf der Basis dieser Daten kann die Bewegungstrajektorie mitgerechnet werden.
Die Herleitung der benötigten Algorithmen beginnt mit dem Aufstellen der erforderlich geometrischen Zusammenhänge des Antriebes. Da sich die Drehzahlen der Antriebsmotoren des Roboters unabhängig voneinander einstellen lassen, kann der Roboter beliebige Bahnradien abfahren, in jedem Fall ist die Kurve ein Segment einer Kreisbahn.

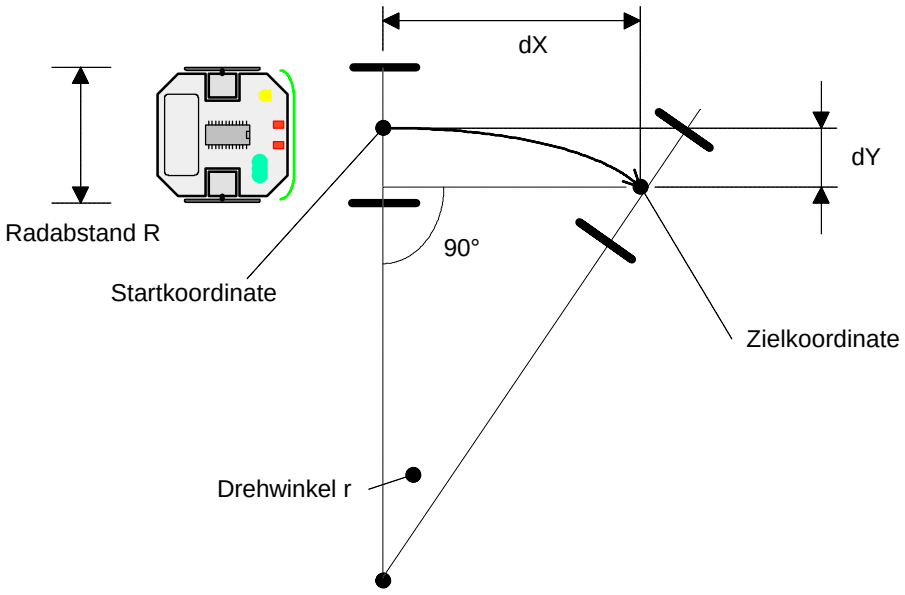



Bild 4.19 Bewegung des Roboters auf einem Kreissegment

Der Kursrechner arbeitet mit einem kartesischen Koordinatensystem. Er muß demzufolge aus der Geometrie der Kreisbahn die neue Kartenposition $P_{Robot}(x,y)$ ermitteln. Für den Kreisbogen gilt:

$$\frac{s_l}{2\pi \cdot r_l} = \frac{s_r}{2\pi \cdot r_r} \text{ mit } s_{l,r} \text{ Wegstrecke Rad links, rechts und } r_{l,r} \text{ Radius.} \quad (4.18)$$

Durch Einsetzen der bekannten geometrischen Abmaße des Radabstandes und Umformen des Ausdruckes errechnet sich der Radius des Kreisbogens des "Robotermittelpunktes" mit der Formel 4.19:

$$r = \frac{R \cdot (s_l + s_r)}{2 \cdot (s_l - s_r)} \ . \quad (4.19)$$

Durch eine Fallunterscheidung muß bei der Implementierung der Sonderfall $s_r = s_l$ (Geradeausfahrt) abgefangen werden. Der Winkel des abgefahrenen Kreisbogens kann ebenfalls durch die Beziehung aus dem Abstand und der zurückgelegten Wegstrecke der Antriebsräder errechnet werden:

$$\alpha = \frac{s_r \cdot 360^o}{2\pi \cdot r_r} = \frac{360^o \cdot (s_l - s_r)}{2\pi \cdot R} \qquad \text{mit R - Radabstand.} \quad (4.20)$$

In [26] wird nun vorgeschlagen, unter Benutzung der Sehne, die durch die Kreisbogen führt, die Koordinatenneuberechnung vorzunehmen. Dieser Vorschlag hat jedoch den Nachteil die jeweilige aktuelle Drehung des Roboters bezüglich des Koordinatensystems als Rechengröße mitzuführen.
Da dieser Drehwinkel bei jedem Interationsschritt der Berechnung erneut in den Algorithmus einfließt und die Berechnung des Drehwinkels auf den fehlerbehafteten Radencoderwerten basiert, bedeutet dies in der Folge eine starke Zunahme des Gesamtfehlers der Berechnung. Das Verfahren wird deshalb modifiziert.
Mit Hilfe eines rechten Winkels, dessen eine Kathede als Fußpunkt auf dem Radius r liegt und die Zielkoordinate schneidet, reduziert sich die Ermittlung der neuen Koordinaten auf Berechnung an dem nun gewonnenen rechtwinkligen Dreieck:

$$\begin{aligned} x_d &= r \cdot \sin\alpha \\ y_d &= r - r \cdot \cos\alpha \end{aligned} \ . \quad (4.21)$$

Bei der Implementierung ist mit Hilfe von Fallunterscheidungen, der Sonderfall $s_r = s_l$ zu finden und über die Vergleiche $s_r > s_l$ bzw. $s_r < s_l$ das Vorzeichen für $x_d$ und $y_d$ zu definieren. Wichtig ist auch die Erkennung von Drehungen auf der Stelle.

Der Kursrechner kann sowohl als Echtzeittask oder als timerunterstützter "Normaltask" laufen. Die Wegstreckenmessung erfolgt im Interrupt der Radencoder. Dort werden die Wegstrecken stückweise vorzeichenbehaftet aufsummiert, d.h. eine Auswerten der Messungen im Koordinatenrechner initialisiert die Meßwerte mit 0.
Die Häufigkeit des Aufrufs des Koordinatenrechners hängt von der gewünschten Genauigkeit der "Streckenmitkopplung" ab. Die Approximation von Kurvenzügen erfolgt als Geradenapproximation. Je öfter gemessen wird, desto kleiner und damit genauer sind die gewonnenen Streckenabschnitte. Die relativ komplizierte Berechnung blockiert oder verlangsamt dafür u.U. wichtigere Tasks. Wird der Zeitraum zwischen den Berechnungen aber zu groß, kann es zu fatalen Fehlberechnungen kommen (Nyquist-Theorem).
Beim Miniroboter wird aus der halben Roboterlänge (50 mm) und der maximalen Geschwindigkeit (50 $mm*s^{-1}$) eine Abtastrate von einer Sekunde gewählt. Das heißt, daß pro Sekunde, die vom System als "wahrer Ort" bekannten Koordinaten aktualisiert werden.
Dieser Zeitraum ist für eine Reihe von speziellen Aufgaben, z.B. Trendschätzungen des Kurses oder für Berechnungen eines "internen Beobachters" relativ grob gewählt. Von Fall zu Fall kann natürlich die Abtastrate verändert werden, besser ist es jedoch, die Meßwerte des Beschleunigungssensors in derartige Berechnungen einzubeziehen [42].

### 4.2.9 Echtzeitdatenkompression

#### 4.2.9.1 Verlustlose Datenkomprimierung mit Huffman-Algorithmen

Im Zusammenhang mit der Bilddatengewinnung bzw. -verarbeitung sind relativ große Datenmengen angefallen. Durch geschickte Wahl des Bildausschnittes konnte die notwendige Speichergröße zur Bildverarbeitung an die Gegebenheiten des Mikrocontrollers angepaßt werden. Zur Speicherung detaillierter Bilder oder Farbaufnahmen reicht der verfügbare interne Speicher nicht aus. Zur besseren Auswertung wäre eine Datenübertragung zum PC sinnvoll.
Problematisch ist hier besonders die begrenzte Übertragungskapazität des Funkmoduls die eine Bewegtbildübertragung unmöglich macht und auch bei Standbildern nur geringe Refreshzyklen zuläßt. Um aus diesem Zwiespalt zu entkommen, muß das Datenvolumen reduziert werden. Als Mittel der Wahl existieren verlustfreie bzw. verlustbehaftete Komprimierungsverfahren. Als verlustfrei sind diejenigen Verfahren zu bezeichen, bei

denen aus den komprimierten Daten das vollständige (und fehlerfreie) Ausgangssignal rekonstruiert werden kann.
Verlustfreie Komprimierungsalgorithmen basieren auf der Häufigkeitsverteilung der Einzelzeichen aus dem Wertevorrat des Übertragungskanals. Im Roboter wird eine Zeichenbreite von 8 Bit verwendet. Der Wertevorrat beträgt demnach:

$$C_{Kanal} = 2^8 = 256. \tag{4.22}$$

In vielen realen Anwendungsfällen ist die Häufigkeitsverteilung der Einzelzeichen stark voneinander abweichend. Ein gutes Beispiel hierfür ist die Häufigkeit des Auftretens verschiedener Buchstaben in einem Text. Im Deutschen ist das mit Abstand häufigste Zeichen der Buchstabe 'e' (17,48 %), gefolgt von 'n', 'i', 'r' and 's' (9,84 %, 7,73 %, 7,54 % and 6,83 %) [43]. Es wäre also sinnvoll, unterschiedlich lange Codes für unterschiedlich häufig auftretende Zeichen einzuführen. Auf dieser Basis arbeitende Verfahren werden als Hufmann-Codes bezeichnet.
Ein geeigneter Algorithmus untersucht die zu komprimierenden Daten auf charakteristische Häufigkeitsverteilungen und erstellt daraus eine Code-Tabelle. Für umgangssprachliche Texte würde dann dem Zeichen 'e' der kürzeste Code und dem Zeichen 'q' (0,02 % Auftretenswahrscheinlichkeit) der längste Code der Tabelle zugewiesen. Bei der Komprimierung werden dann anstelle der Orginaldaten die "umkodierten" Informationen übermittelt. Sender und Empfänger benötigen zur korrekten Dekomprimierung die gleiche Code-Tabelle. Je nach Häufigkeitverteilung der Daten im Text sind variierende Komprimierungsraten zu erwarten.
Ist die Häufigkeitsverteilung beliebiger Daten unbekannt, muß die Tabelle zur Laufzeit der Komprimierung erstellt und vor dem Dekomprimieren an den Empfänger übermittelt werden. Im genannten Beispiel von 8-Bit-Daten umfaßt diese Tabelle 256 Einträge. Damit das Übertragungsverhältnis zwischen komprimierten Daten und Codetabelle nicht zu ungünstig ausfällt, müssen ausreichend große Datenmengen auf ihre Häufigkeitsverteilung untersucht werden. Problematisch sind zudem wechselnde Häufigkeiten innerhalb des zu übermittelnden Datenstromes. Hier müssen dann laufend neue Codetabellen errechnet und übertragen werden. Vollends aussichtslos ist die Komprimierung von gleichverteilten Daten mit diesem Verfahren.
Neben der charakteristischen Häufung von Zeichen innerhalb bestimmter Datensätze gibt es auch Datenströme, deren sequentielle Abfolge eine starke Korrelation der aufeinanderfolgenden Daten aufweist. Bilddaten beinhalten oftmals Bereiche mit geringer Werteänderung. Der CMOS-Sensor der Roboterkamera liefert schwarz-weiß Bilder mit 224 Graustufen. Mehr als 64 Graustufen kann das menschliche Auge nicht voneinander

trennen. Es ist also relativ einfach möglich, die unteren Intensitätswerte auszumaskieren und dann mit einer Run-Lengh-Codierung die Bilddaten zusammenzufassen (RLE - *run lenght encoding*).

### 4.2.9.2 Adaptive Differential Pulse Code Modulation (ADPCM)

Abweichend von den verlustfreien Komprimierungen werden bei der verlustbehafteten Kompression Abweichungen des Orginalsignales nach der Dekompression in Kauf genommen. Ein sehr effektives Verfahren hierzu ist die *Adaptive Differential Pulse Code Modulation* (ADPCM). Grundlage des Verfahrens ist auch hier die Korrelation zwischen aufeinanderfolgenden Daten.

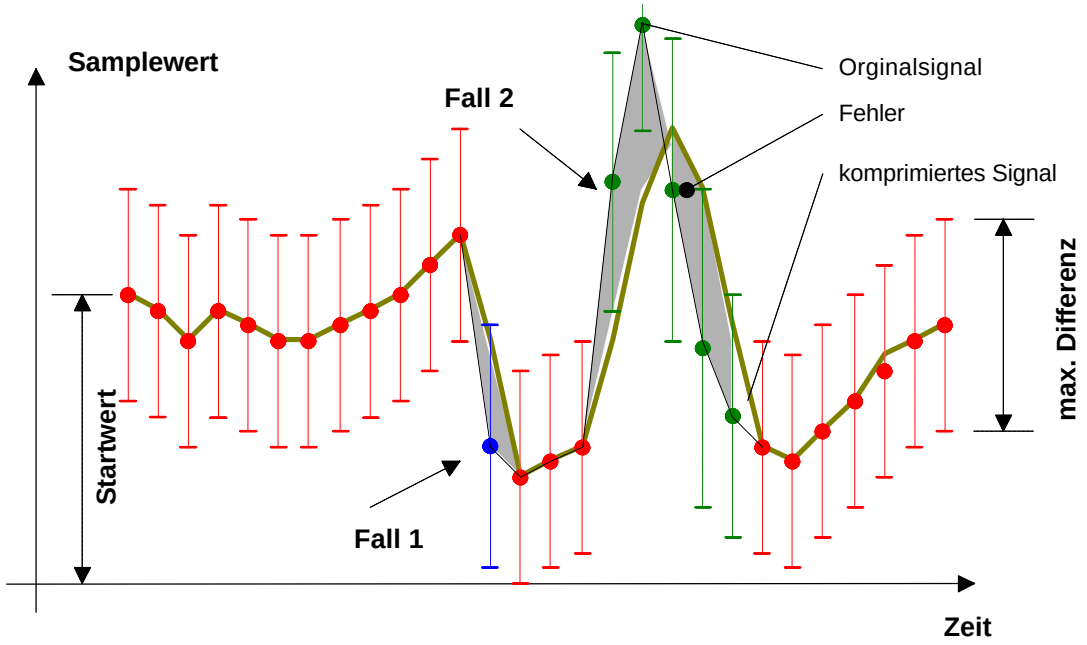



Bild 4.20 grafische Darstellung des ADPCM Verfahrens

Die Differenz aufeinanderfolgender Daten ist oftmals sehr viel geringer als der jeweilige Absolutwert. Damit verringert sich der erforderliche Wertevorrat der zu übertragenden Zeichen und folglich auch die zu übertragende Datenmenge. Wäre in einem Beispielsignal die Differenz immer innerhalb des Bereiches von -8...7 (16 Werte), genügte eine 4-Bit Übertragung anstelle der sonst notwendigen 8-Bit.
Reale Signale gehorchen derartigen Bedingungen nur zeitweise. Die Abbildung 4.18 zeigt ein "reales Signal". An den mit Fall 1 und 2 gekennzeichneten Stellen ist die Differenz zweier Samplewerte größer als der Wertebereiche (max. Differenz) des vereinbarten 4-Bit Wertes. Das komprimierte Signal kann dem Orginalsignal nicht mehr folgen. Es

entsteht ein Fehler. Im ersten Fall würde dieser nach wenigen weiteren Abtastungen wieder verschwinden, da die folgenden Differenzen wieder kleiner als der maximale Differnzwert sind.
Die Fehler steigen jedoch über alle Maßen an, wenn kräftige Sprünge auftreten. Diese können nur durch größere zugelassene Differenzwerte abgebildet werden. Damit jedoch der Wertevorrat von 16 Zeichen (= 4-Bit) weiter ausreicht, wird der 4-Bit-Wert als Zeiger auf eine Tabelle interpretiert.
Der Kompressionsalgorithmus wird um adaptive Fähigkeiten erweitert. Immer dann, wenn der Grenzwert des Differenzwertes erreicht ist, werden in der Indextabelle die Abstufungen vergrößert bzw. verkleinert. Codierer sowie Decodierer verhalten sich invers zueinander, damit funktioniert das Verfahren in beiden Richtungen.
Der Adaptionsalgorithmus benötigt zur Anpassung an variable Differenzen einen Prädiktor (Vorhersagewert, Schätzwert).

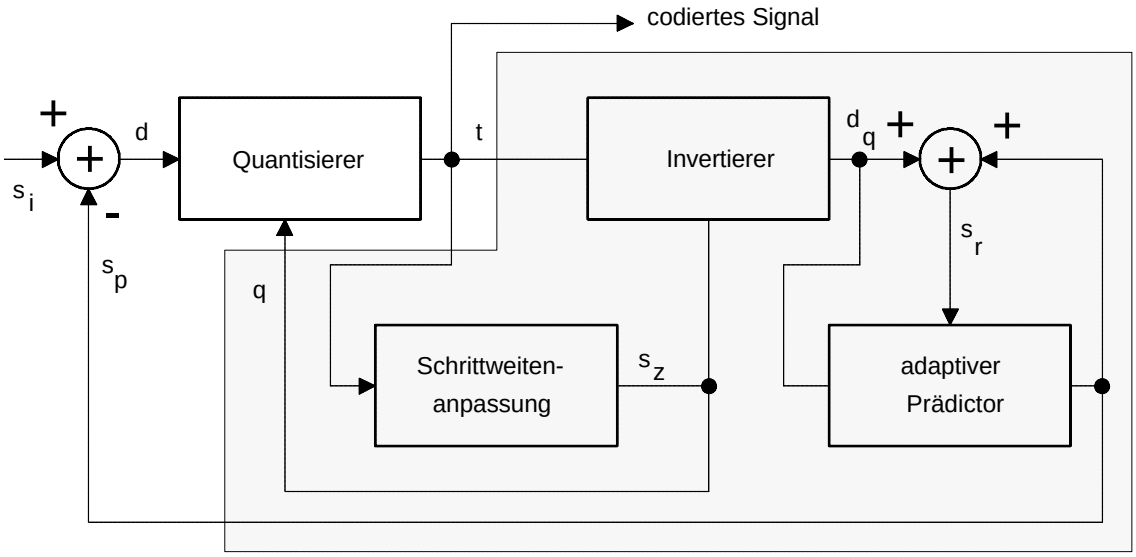



Bild 4.21 ADPCM Encoder/Decoder

Die Differenz *d* wird aus dem Vorhersagewert $s_p$ und dem exakten Eingangswert $s_i$ gebildet. Daraus entsteht der quantisierte Encoderwert *t*, der sowohl als codiertes Signal anzusehen ist, als auch als Eingangswert zur Schrittweitenanpassung und zur Bildung des nächsten Vorhersagewertes $s_p$, dient. Der als Invertierer gekennzeichnete Block ist als inverser Quantisierer, der aus *t* und der eingestellten Schrittweite $s_z$ (step size) den Wert $d_q$ (inverse Quantisierungswerte) errechnet. Dieser grau hinterlegte Block stellt somit den Decoder dar. Für die Implementiereung bedeutet dies, daß große Teile der erforderlichen Software von Encoder und Decoder gemeinsam genutzt werden können.

ADPCM-Algorithmen sind relativ einfach zu implementieren und liefern schnelle und genügend brauchbare Kopmprimierungsraten. Ein Beispiel für 8-Bit Conroller ist in [27] zu finden. Von großem Vorteil ist auch die konstante Komprimierungsrate. Für den Robotereinsatz werden zwei Funktionen *bAdpcmCoding()* sowie *bAdpcmDecoding()* bereitgestellt. Die Komprimierungsrate beträgt 50%.

### 4.2.9.3 Block Truncation coding

Einer der wesentlichsten Nachteile von ADPCM Verfahren bei der Bildkomprimierung ist die ausschließliche Einbeziehung des Vorgängers in die Datenkomprimierung. Bilder sind jedoch zweidimensionale Objekte und als solche ist eine Korrelation von Intensitätswerten nicht nur zwischen Vorgänger und Nachfolger innerhalb einer Bildzeile, sondern auch zwischen den jeweiligen benachbarten Bildpunkten zu erwarten.
Diese Ähnlichkeit der Intensitätswerte benachbarter Bildpunkte ist die Basis des *Block Truncation Codings (BTC).* Das Gesamtbild wird in nicht überlappende Areale der Größe *n* * *m* Pixel zerlegt. Als Intensitätswerte wird 17 für schwarze und 240 für weiße Bildpunkte definiert. Für jedes Areal werden der Mittelwert $\mu$ und die Standardabweichung *o* errechnet:

$$\mu = \frac{1}{n \cdot m} \sum_{i=1}^{n} \sum_{j=1}^{m} Y_{i,j} \quad , \tag{4.23}$$

$$\sigma = \sqrt{\frac{1}{n \cdot m} \sum_{i=1}^{n} \sum_{j=1}^{m} (Y^2 - \mu^2)} \quad . \tag{4.24}$$

Mit Hilfe des Mittelwertes wird das Pixelareal in ein Binärbild zerlegt, welches nach folgender Vorschrift gebildet wird:

$$B_{i,j} = \begin{cases} 1 ... \textit{für } Y_{i,j} \leq \mu \\ 0 ... \textit{sonst} \end{cases} \quad . \tag{4.25}$$

Daneben werden noch zwei Intensitätswerte für die dunkleren Pixel *a* sowie für die helleren Pixel *b* benötigt:

$$a = \mu - \sigma \sqrt{\frac{p}{q}} \quad , \tag{4.26}$$

$$b = \mu + \sigma\sqrt{\frac{q}{p}}\ . \qquad (4.27)$$

Dabei ist $p$ die Anzahl der Pixel, die heller als der Mittelwert $\mu$ sind, und $q$ die Anzahl derjenigen Pixel dunkler als $\mu$.

### 4.2.9.4 Beispielrechnung für BTC-Bildkomprimierung

Für den Spezialfall der manuellen Fernkontrolle der Roboter durch einen Operator oder bei schnellem Szenarienwechsel und damit verbundener Bildverarbeitung am PC soll die Bildübertragung so schnell wie möglich erfolgen.
Die Datenrate des Funkkanals ist mit max. 115200 Baud vorgegeben. Als Bildformat werden eine Bildgröße von 176 x 144 Bildpunkten (QCIF-Format) als ausreichend angesehen. Die Bilddatenrate soll 1 Frames pro Sekunde erreichen.

Unkomprimiertes Orginalbild

Dekomprimiertes Bild aus einer 8 x 8 Blockkomprimierung

Der Bildausschnitt zeigt die Blockartefakte des Komprimierungs-verfahrens

Bild 4.22 Beispiel für Bildkomprimierung mittels BTC

Ein komplettes QCIF-Bild hat eine Größe von 25344 Byte. Bei 115200 Baud würde die Übertragung eine Zeitdauer von ca. 2,2 Sekunden in Anspruch nehmen. Das ist um den Faktor 2,2 zu groß.
In der Abbildung 4.20 ist ein Beispielbild, das mit einem Pixelareal von 8x8 Pixel komprimiert wurde, zu sehen. Die Qualität wird als ausreichend für die Fernkontrolle durch einen menschlichen Operator eingeschätzt. Inwiefern die Bildkompression eine nachträglich Bildverarbeitung durch einen Rechner beeinflußt ist nicht untersucht worden.

Der Kamerasensor kann so programmiert werden, daß das QCIF-Bild frei auf einen Bereich des Gesamtbildes (Ausschnitt aus der 352 x 288 Pixel-Matrix) positioniert werden kann. Bei zwei Bildern pro Sekunde dauert das Abtasten einer Bildzeile ca. 3,4 ms. Da je acht Zeilen zur Berechnung nötig sind, wird das QCIF-Bild per DMA (*direct memory access*) mit jeweils acht Zeilen im Block eingelesen. Das Einlesen eines kompletten Blockes erfordert dann ca. 27 ms und belegt 1408 Byte Speicherplatz. Aus der Komprimierungsrate von 6,4 : 1 errechnet sich ein zu sendender Datenstrom von 220 Byte pro Block. Zuzüglich einiger Markierungsbytes müssen dann 230 Byte in 27 ms übertragen werden. Das Funkmodem benötigt dazu bei 115200 Baud rund 20 ms. Da Bildeinlesen und die Bildübertragung per DMA erfolgt, steht, mit Ausnahme der Echtzeittasks im Hintergrund, beinahe die vollständige Prozessorleistung zur Verfügung.

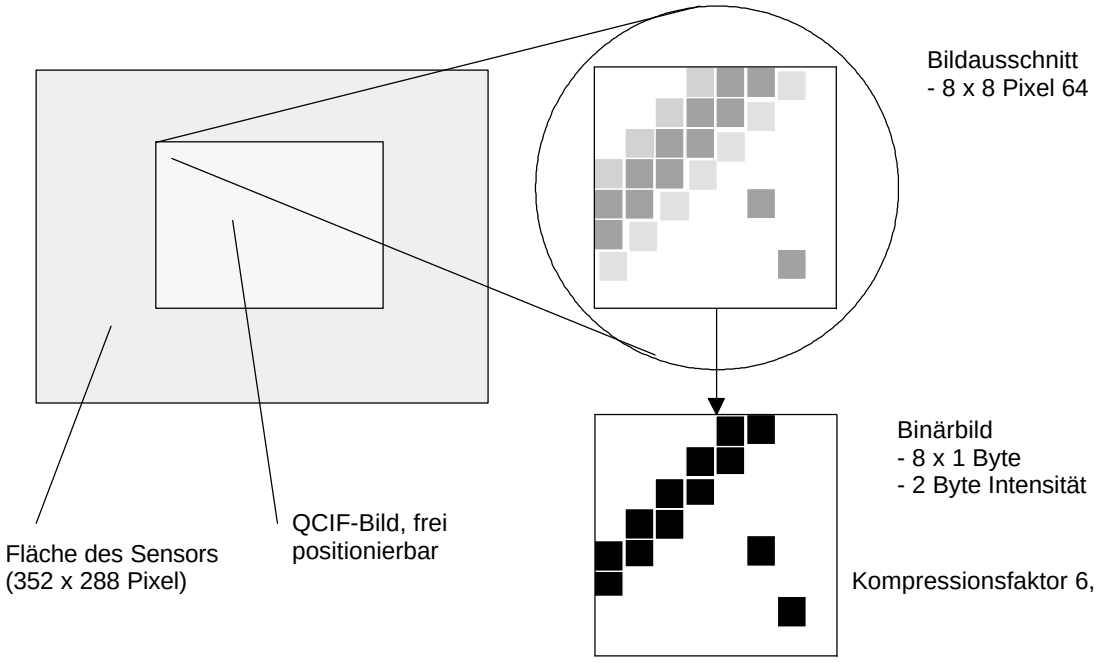



Bild 4.23 Echtzeitbildkomprimierung beim Roboter MauSI 2

Für die Implementierung auf dem Mikrocontroller werden die unhandlichen Formeln 4.24, 4.26 und 4.27 umgearbeitet. Die Berechnung der Standardabweichung wäre in dieser Form viel zu aufwendig (Wurzel aus der Summe von Quadraten) und würde unnötig Rechenzeit belegen. Die Gleichungen 4.26 und 4.27 stellen im wesentlichen den Intensitätsmittelwert der jeweiligen Pixelmenge, die den Mittelwert des Gesamtblockes bildet, dar. Für die Berechnung der Parameter a und b wird während der Binarbildgenerierung der Intensitätswert der betreffenden Punktmenge berechnet und über die Punktanzahl gemittelt.

```
/*************************************************************************/
/* Subroutine:  BTC CODING                                               */
/* Autor:       Jens Altenburg                                           */
/*************************************************************************/
stCodedBlock bBtcCompress ( stPixelBlock stPixel )
    {
    byte i,j, bMittel;
    byte p = 0;                                 /* Pixelzähler        */
    word w = 0;                                 /* Hilfsvariable      */
    word a = 0;                                 /* Intensität A       */
    word b = 0;                                 /* Intensität B       */
    stCodedBlock stCompress;
    for (i = 0; i < nAreaSize; i++)
        {
        for (j = 0; j < nAreaSize; j++)
            {
            w += stPixel.abBlock[i][j];
            }
        }
    bMittel = (byte)(w >>= 6);                   /* Mittelwert bilden */
    for (i = 0; i < nAreaSize; i++)
        {
        for (j = 0; j < nAreaSize; j++)
            {
            if(stPixel.abBlock[i][j] <= bMittel)
                {
                stCompress.abBinaer[i] |= (1 << j); /* Binärpixel setzen */
                a += stPixel.abBlock[i][j];
                p++;
                }
            else
                {
                stCompress.abBinaer[i] &= ~(1 << j);/* Binärpixel löschen */
                b += stPixel.abBlock[i][j];
                }
            }
        }
    if(p > 0)
        {
        stCompress.bA = (byte)(a / p);
        stCompress.bB = (byte)(b / ((nAreaSize * nAreaSize) - p));
        }
    else
        {
        stCompress.bA = 0;
        stCompress.bB = (byte)(nAreaSize);
        }
    return stCompress;
    }
```

Anhand dieses Beispieles eines Kompressionsverfahrens soll die enge Verzahnung von Implementierung und Ausführungsgeschwindigkeit der gefundenen Lösung verdeutlicht werden.

Eingangs wurde für das Einlesen des zu komprimierenden Blocks ein Zeitbedarf von ca. 27 ms ermittelt. Per DMA werden dann zwei dieser "Bildblöcke" abwechselnd mit Daten gefüllt. Der jeweils gerade geladene Block muß nun komprimiert und übertragen werden. Der Datentransfer zum Funkmodul erfolgt ebenfalls per DMA. Der Steuercontroller muß nun vor dem Start der Funkübertragung einen entsprechenden Vorlauf der

komprimierten Daten schaffen, damit sich Komprimierung und Datenübertragung nicht ins Gehege kommen. Die Grafik 4.22 zeigt den entstehenden zeitlichen Zusammenhang.

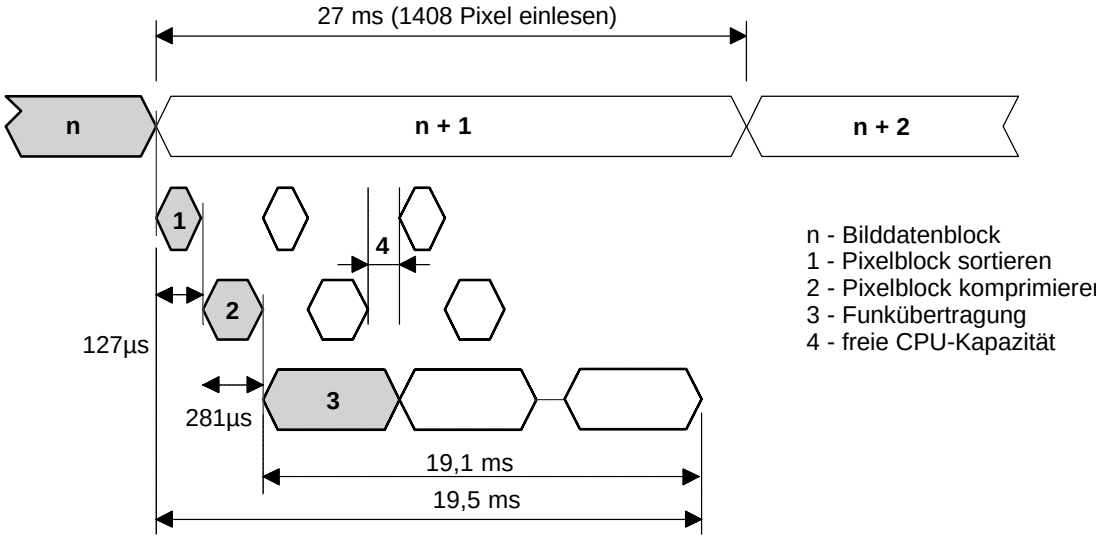



Bild 4.24 Zeitliche Abfolge von Bilddatengewinnung, Kompression und Übertragung

Ein erster Test des implementierten Beispiels ergab eine Rechenzeit von 648 µs für den Kompressionsalgorithmus. Hinzu kommt noch eine Zeitspanne von 536 µs für die Datenaufbereitung als 8x8 Byte Pixelblock (26 ms Gesamtzeit pro Block). Das würde prinzipiell für eine Echtzeitübertragung ausreichen. Allerdings läge die Prozessorlast für die Bildübertragung bei ca. 96 %. Die verbleibenden 4 % Rechenzeit dürften für die Grundsteuerung des Roboters und die Echtzeittasks kaum noch ausreichen.
Eine Analyse des vom Compiler generierten Codes offenbart die Schwachstellen. Beim Aufruf der Funktion *bBtcCompress()* wird der vorsortierte Pixelblock übergeben und beim Beenden der komprimierte Block zurückgeliefert. Beide Operationen sind mit aufwendigen Kopiervorgängen im Speicher des Controllers verbunden. Die Routine wird deswegen auf Zeigeroperationen bei den Speicherzugriffen umgestellt. Anstelle der Blöcke erhält der Algorithmus nur die Verweise auf die Start- und Zieladressen übergeben. Der Aufruf erfolgt nun als *bBtcCompress(&PixelBlock, &BinaerBild)* und vermeidet sinnlosen Datentransfer. Ähnlich wird auch die Datenaufbereitung beschleunigt. Der Rechenzeitbedarf sinkt auf 9,1 ms pro Bildblock, d.h. die Prozessorlast liegt bei jetzt bei ca. 35 % (Darstellung in Bild 4.22 ist nicht maßstabgerecht).
Die genannten Zahlenwerte lassen vermuten, daß mit einem gewissen Qualitätsverlust, eine online Datenübertragung vom Roboter zum Steuer-PC möglich ist. Der wesentliche Flaschenhals ist die Funkdatenübertragung und die Ankopplung über die serielle Schnittstelle an den PC. Das Funkmodem würde zwar eine Datenrate von 160KBaud gestatten,

mit 115,2 KBaud ist die Leistung des RS232-Interfaces am PC jedoch am Anschlag. Deutlich leistungsfähigere Lösungen zeichnen sich erst mit der Änderung des benutzten Frequenzbandes, 2.4 GHz statt 433 MHz und anderer Funkverfahren, ab. Bessere Systeme leisten ca. 1 MBaud, so daß auch Bewegtbildübertragung möglich wäre.

## 4.3 Softwareentwicklung in *Embedded-Control*-Systemen

Die Effizienz von Softwareentwicklungen wird maßgeblich von den verfügbaren Entwicklungstools beeinflußt. Charakteristisch bei der Softwareentwicklung von *Embedded-Control-Systemen* ist dabei die Trennung von Softwareentwicklung und Softwaretest.
Kennzeichnend für ein solches System ist das (fast) vollständige Fehlen von Ein-/Ausgabegeräten. Normale Personalcomputer sind mit einer Vielzahl von Gerätschaften zur Kommunikation mit dem Bediener, z.B. Tastatur, Bildschirm, Drucker, etc., ausgerüstet. Kommunikationsschnittstellen für ein Mensch-Maschine-Interface sind bei Robotern eher selten, es sei denn sie sind für den Dialog mit Menschen konzipiert. Übliche Systeme sind vielmehr als selbstständige autonome Agenten angelegt.
Diese "Sparsamkeit" hinsichtlich verfügbarer Schnittstellen ist insbesondere während des Programmentwurfes problematisch. Softwaredesign am PC bietet dem Programmierer vielfältige Möglichkeiten, seine Algorithmen einzugeben (Sourcecode-Editor), syntaktische bzw. schwerwiegende logische Fehler automatisch zu finden (Compilerläufe) bzw. letztendlich auf Sourcecode-Level Variable anzusehen, zu setzen oder das Programm schrittweise abzuarbeiten (Debugging).
Eine derart komfortable Entwicklungsumgebung ist bei Mikrocontrollerentwicklungen oft sehr kostspielig, da sie den Einsatz eines speziellen Emulators voraussetzt. Ein solcher Emulator ist ein Gerät, das alle Funktionen des Steuercontrollers nachbildet und dem Programmierer den Zugriff auf die internen Ressourcen, z.B. Speicher bzw. Register des Steuercontrollers, erlaubt.
Neben der begrenzten Verfügbarkeit von Emulatoren (Kostenpunkt einige Tausend Euro) ist diese Art der Fehlersuche u.a. mechanisch recht umständlich. Immerhin muß ein vielpoliges Gehäuse, im Fall von MauSI 2 ist dies ein 100-poliges Package mit 0,5 mm (!) Pinabstand, über einen Adapter an den Emulator angeschlossen werden.
Wesentlich einfacher ist die Nutzung interner Debug-Schnittstellen des Mikrocontrollers. Wie bereits im Abschnitt 4.2.3 kurz angerissen, kann der M16C mit Hilfe einer seriellen Verbindung von einem PC aus "debugged" werden.

Die Softwareentwicklung teilt sich deswegen in zwei Abschnitte auf. Zur Programmeingabe und zum Finden syntaktischer Fehler dient ein Editor zusammen mit dem Compiler und einer Projektverwaltung auf dem PC.
Über den Editor kann in gewohnter Weise der Programmcode eingegeben werden. Spezielle Features wie "Syntax-Highlighting", d.h. die farbliche Hervorhebung von Schlüsselworten, erhöhen den Bedienkomfort und erleichtern die Lesbarkeit der Programme. In den Editor integriert sind Projektverwaltung und Compiler. Mit der Projektverwaltung wird der Zusammenhang der einzelnen Module und deren Speicherzuweisung kontrolliert.
Besonders die Speicherverwaltung mit der damit verbundenen Zuweisung der Variablen an verschiedene Adressen im Adressraum des Controllers ist für PC-Programmierer anfangs oft verwirrend. Die richtige Zuweisung der verschiedenen Speichertypen ist für die Effizienz und Ausführungsgeschwindigkeit des erzeugten Programmcodes zumeist von ausschlaggebender Bedeutung. Während beim PC oft das gesamte Programm vor der Ausführung komplett in den Arbeitsspeicher (RAM) geladen wird, ist ein solches Vorgehen bei *Embedded Controllern* undenkbar. Bereits während der Programmentwicklung muß der Programmierer sich Gedanken über die Zuteilung des Speichers an die einzelnen Programmodule machen. Da es bei größeren Systemen oft unmöglich ist, sämtliche Ressourcen manuell zu überwachen, helfen spezielle Tools, wie z.B. Stackviewer, dabei rechtzeitig Engpässe zu erkennen.
Speicherüberlappungen, d.h. Speicherbereiche, die versehentlich von mehreren Modulen gleichzeitig genutzt werden, findet (oft) der Linker.
Der zweite Teil der Softwareentwicklung findet direkt im Zielsystem statt. Die Abbildung 4.22 zeigt einen Bildschirmausdruck während einer typischen Debugsession.
Der eigentliche Debugger besteht aus zwei Komponenten. Ein Teil läuft als Anwendungsprogramm auf dem PC. Dieser Teil beinhaltet im wesentlichen die komfortable Visualisierung des Programmcodes und der angezeigten Variablen. Das Programm greift dazu auf bestimmte Symboldateien, die während des Compilerlaufes entstehen, zurück.
Der andere Programmteil ist als sogenannter Monitor in einem "unbenutzten" Speicherbereich des Controllers abgelegt. Der Monitor kontrolliert die schrittweise Programmabarbeitung im Zielsystem (*Adressmatch-Interrupt*) und liefert die Inhalte der vom Anwender angeforderten Variablen an das PC-Anzeigeprogramm zurück.
Monitor und Visualisierung kommunizieren über eine serielle Schnittstelle des Controllers. Der Monitor darf nur minimale Ressourcen im Zielsystem, RAM, ROM bzw. Rechenzeit, belegen.

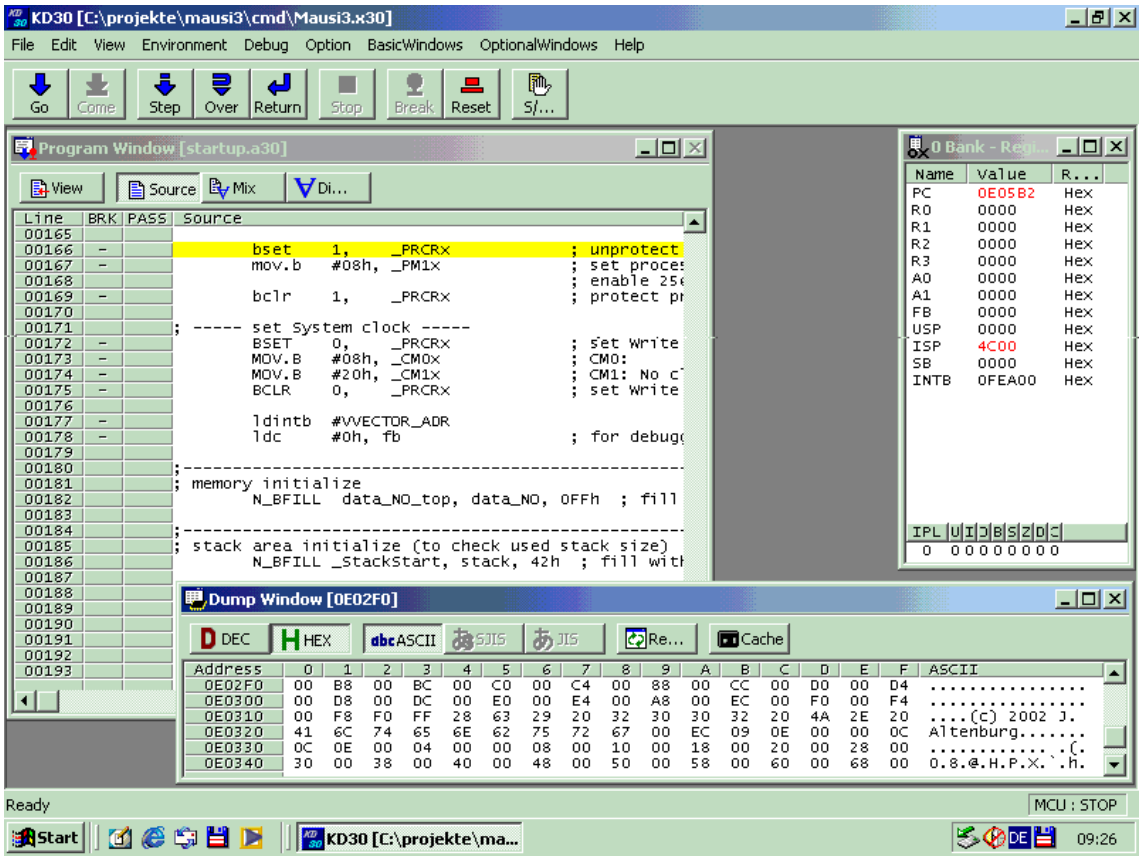



Bild 4.25 Bildschirmansicht während des Debuggens

Für den Anwender hat dies den Anschein, als ob er direkt die Ressourcen des M16C kontrollieren kann. Dieser Anschein stimmt nur teilweise. Selbstverständlich benötigt der Monitor Rechenzeit des M16C, die dann der Applikation fehlt. Damit verlangsamt sich die Programmausführung geringfügig. Da der Monitor auch Interrupts belegt, ist ein minimales Verständnis für seine Funktion für ein effizientes Debuggen unerläßlich. Trotz einiger Nachteile, ist die *In-Circuit-Emulation* eine effektive, kostengünstige Alternative zum Emulator.

# 5 Bahnführung und Hindernisvermeidung

## 5.1 Methoden der Bewegungsplanung

Wesentliches Kennzeichen von Robotersteuerungen ist die mehr oder minder ausgeprägte Fähigkeit zur Bahnplanung mittels Sensordateninterpretation. Weltmodellbasierte Steuerungen ermöglichen anhand der Sensordaten und deren Abgleich mit vorliegenden Kartenmaterial oft eine Selbstlokalisation des Systems.
Es existieren eine Reihe möglicher Planungsalgorithmen. Die nachfolgend kurz erläuterten Verfahren werden in der Literatur, z.B. [32], [33] besonders häufig verwendet.

### 5.1.1 Bewegungsplanung auf der Basis Kartendaten

Dieses Verfahren beruht auf dem Versuch der möglichst exakten Generierung einer Straßenkarte (*roadmap*). Das Straßennetz dieser Karte beinhaltet alle möglichen Pfade die in einer mit Hindernissen bestückten zweidimensionalen Ebene möglich sind. In der ersten Näherung wird dabei nicht zwischen begehbaren bzw. nicht begehbaren Pfaden unterschieden.

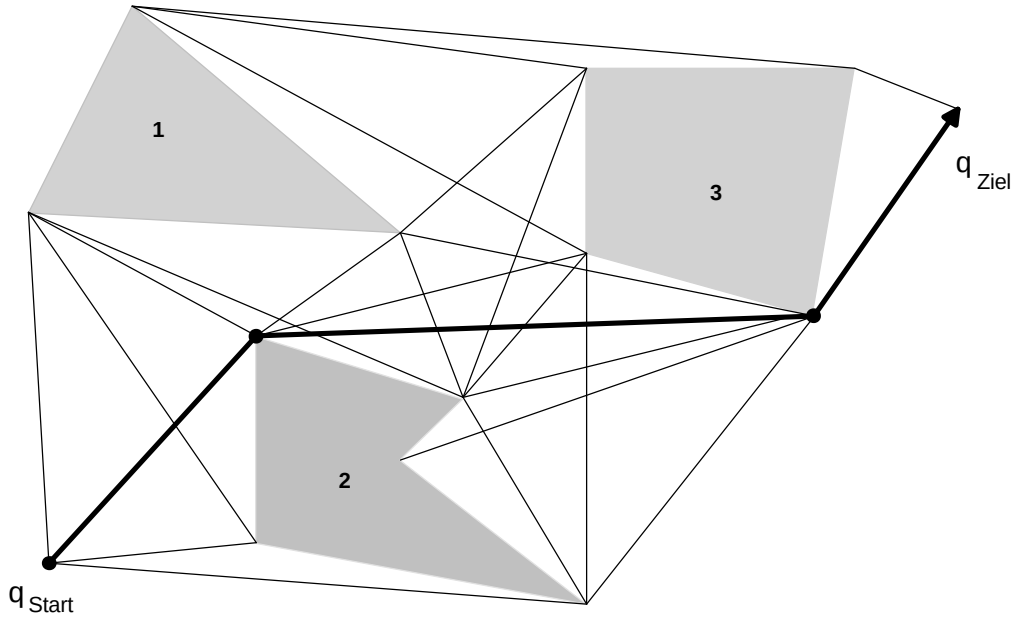



Bild 5.1 Roadmapgenerierung mittels *Visibility Graph Method*

Ein Verfahren zur Erzeugung der Roadmap besteht in der Anwendung der *Visibility Graph* Methode. Die Abbildung 5.1 verdeutlicht das Vorgehen. Alle Ecken der

Hindernisse werden unter der Bedingung kein anderes Hindernis zu kreuzen, durch Linien miteinander verbunden. Weitere Linien verbinden Start- und Endposition des Roboters mit den erreichbaren Ecken der Hindernissse.
Es ensteht ein Graph G mit der Menge aller Knoten V und der Menge der Linien E:

$$G = (V,\ E)\ . \tag{5.1}$$

G kann nun nach einem geeigneten Pfad von $q_{Start}$ nach $q_{Ziel}$ durchsucht werden. Die Verfahrensweise hat jedoch einige Nachteile. Dabei wird der Nachteil einer nur zweidimensionalen Pfadplanung oft hingenommen. Schwerer wiegt das Problem der sequentiellen Pfadextraktion (erst Graph bilden, dann Pfad suchen), das beim Auftauchen bewegter Hindernisse den gefundenen Pfad sofort zu Makulatur werden läßt. Hier schafft nur die sehr aufwendige ständige Pfadneuplanung Abhilfe.
Je nach Struktur der zweidimensionalen Hindernisse kann die Bestimmung des Graphen bei zunehmend komplexer werdenden Polygonen extrem umfangreich werden (Anzahl der Linien und Knoten steigt). In [32] wird alternativ deswegen ein *Voroni-Graph* verwendet, um die Datenmengen zu reduzieren. Dabei werden Pfade definiert, deren minimale Abstände zwischen mindestens zwei Hindernissen identisch sind.

### 5.1.2 Wegplanung durch Zellteilung

Die Methode zerlegt den Arbeitsraum in Zellen (engl. *cell decomposition*). Die Zellgrenzen werden wiederum anhand der Eckpunkte der Hindernisse definiert.

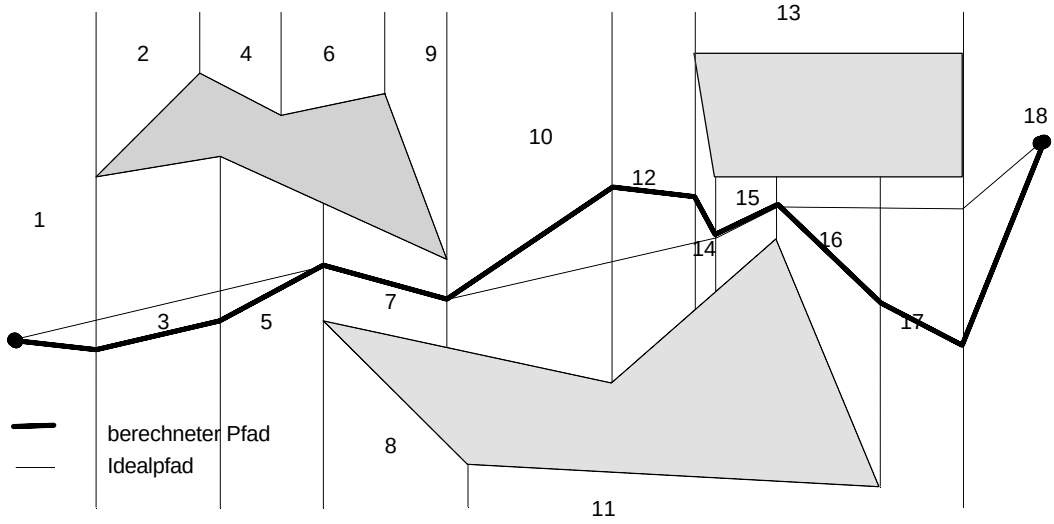



Bild 5.2 exakte Zellteilung für die *Cell Decomposition* Annäherung

Der gefundene Pfad würde im Beispiel dem Graphen {1-3-5-7-10-12-14-15-16-17-18} entsprechen. Die jeweiligen Mittelpunkte der Linien bilden die Knoten des Pfades.
Ein Blick auf das Beispiel zeigt, daß durchaus mehrere Pfade vom Start- zum Zielpunkt existieren können. Mehrdeutigkeiten zu extrahieren, bzw. einen optimalen Pfad zu finden, obliegt dann weiteren Analyseschritten. Gleichzeitig können unnötige Abweichungen von der Ideallinie des Fahrweges minimiert werden.
Das Beispiel im Bild 5.2 zeigt die exakte *Cell Decomposition*. Aus dem Beispiel wird jedoch ebenfalls ersichtlich, daß die Zerlegung in Zellen stark redundant ist. Es wäre nicht zwingend notwendig, den gesamten Arbeitsraum in Zellen zu zerlegen.
Eine Methode zur Reduktion des erforderlichen Aufwandes besteht in der Anwendung der annähernden Zellteilung.

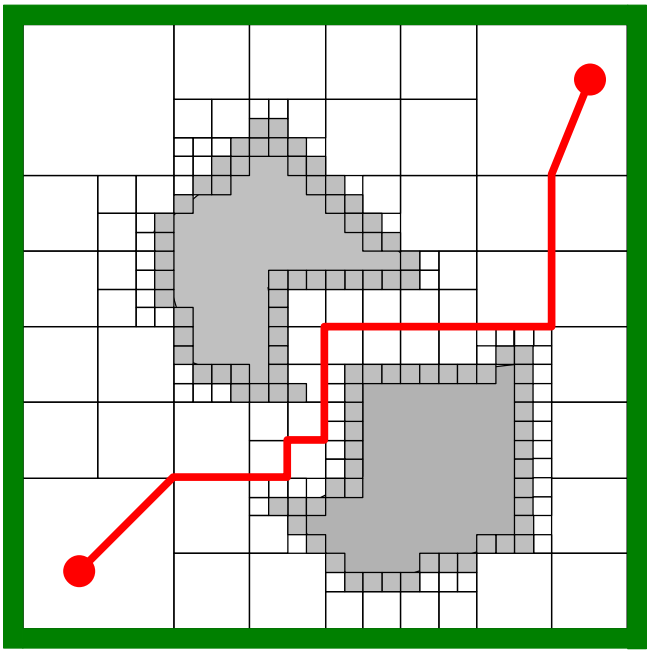


Bild 5.3 annährende rekursive Zellteilung

Der Arbeitsbereich wird solange in Rechtecke unterteilt, bis diese eindeutig entweder einem Hindernis oder unbesetztem freien Raum zugeordnet werden können. Es ist nicht erforderlich, den gesamten Raum zu unterteilen, die Teilung kann abgebrochen werden, wenn die Auflösungsgrenze erreicht bzw. ein Pfad gefunden wurde. Der gefundene Pfad verläuft entlang der Zellgrenzen, die sicher frei von Hindernissen sind.
Wird die Zellgröße in geeigneter Weise an die Abmaße des Roboters geknüpft, kann auf eine separate "Hindernisvergrößerung" (siehe Abschnitt 5.1.4) verzichtet werden.

### 5.1.3 Streckengenrierung durch Potentialfelder

Die zell- oder linienorientierten Verfahren liefern einen Graphen als Ergebnis ihrer Pfadplanungen. Einen gänzlich anderen Ansatz verfolgt die Wegplanung mit Hilfe von Potentialfeldern (engl. p*otential field*). Der Roboter wird als Teilchen eines Potentialfeldes angesehen. In Abhängigkeit vom Ort innerhalb des Arbeitsbereiches wird dieses Teilchen unterschiedlichen Kräften ausgesetzt.

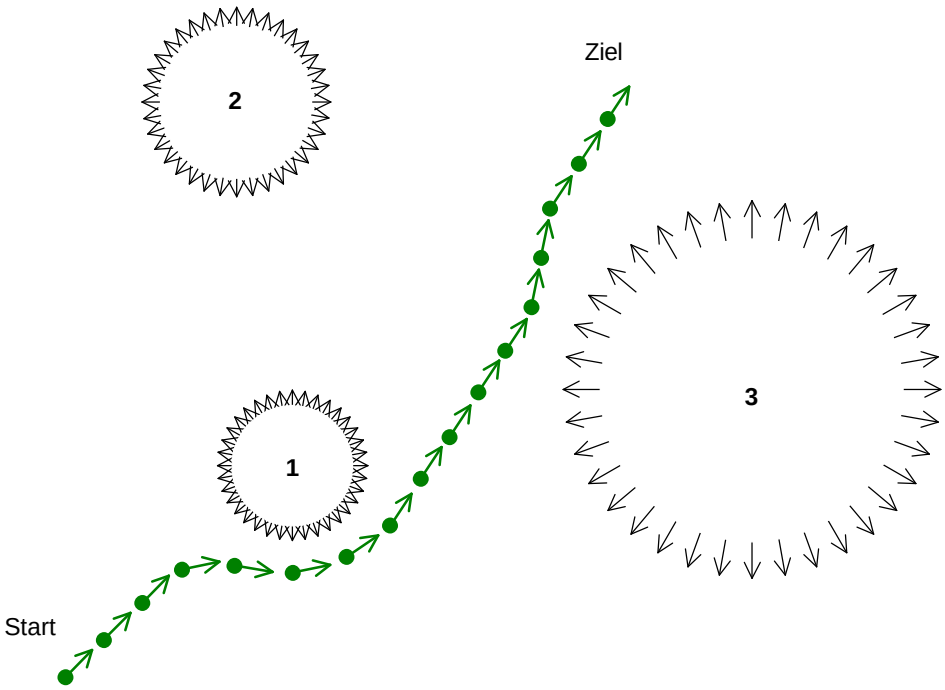



Bild 5.4 Hindernisse in Potentialfelddarstellung

Die Abbildung deutet die Darstellung der Hindernisse als Störungen im Potentialfeld an. Ohne Störung würde der Roboter in gerader Linie vom Start- zum Zielpunkt durchlaufen. Durch die Hindernisse wird der Kurs des Roboters vom Hindernis weggeführt. An jeden Punkt des Arbeitsbereiches überlagern sich die Kraftvektoren zu einer resultieren Kraft, die als Aufschaltgröße den Kurs entsprechend korrigiert.
Eine besonders anschauliche Vorstellung ist die einer Kugel, die in einer entsprechend geformten geneigten Ebene, vom Startpunkt (höchster Punkt) zum Ziel (tiefster Punkt) hinabrollt. Die Hindernisse entsprechen dann Erhöhungen der Ebene, so daß die Kugel um dieser herum abrollt.
Der wesentlichste Nachteil dieses Modells ist das mögliche Auftreten lokaler Minima, d.h. es sind Fälle denkbar, in denen die Kugel vor Erreichen des Zieles zum Stehen

kommt, weil die umgebenden Potentialkräfte einen Halt erzwingen. Solche Fälle sind insbesondere dann ärgerlich, wenn andere Pfadplanungen Lösungen fänden.
Dennoch erscheint der Ansatz Hindernissen Potentiale zuzuordnen und im Verlaufe der Bewegung die Pfadplanung zu optimieren so erfolgversprechend, daß hier versucht werden soll, die offenkundigen Nachteile (Deadlock im lokalen Minimun) auszuschließen.

### 5.1.4 Künstliche Hindernisvergrößerung zur Extraktion realer Pfade

Alle diese Verfahren (mit Ausnahme der approximierten Zellteilung) berücksichtigen nicht die prinzipielle Begehbarkeit eines potentiellen Pfades. Um hier nicht laufend Abstandsberechnungen durchführen zu müssen, ist es sinnvoll, die Hindernisse um die Abmaße des Roboters zu vergrößern.

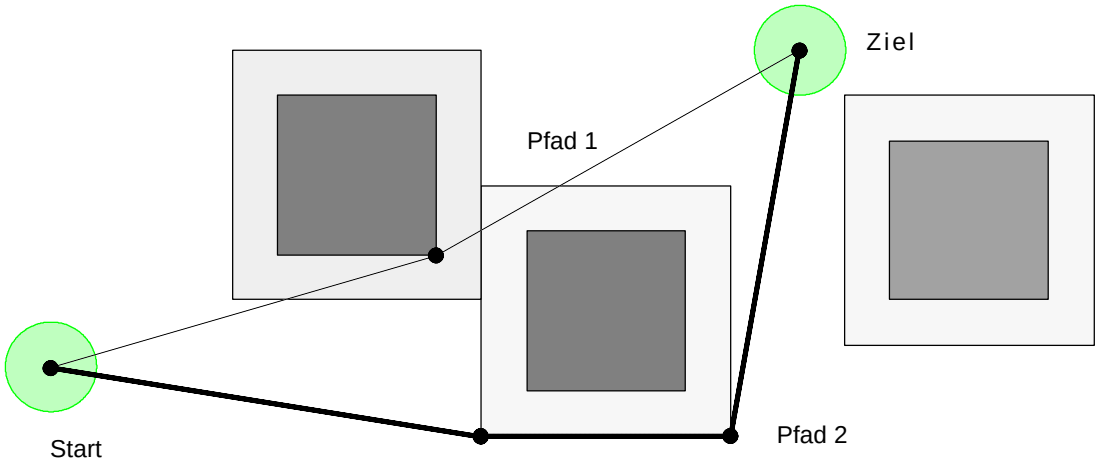



Bild 5.5 "Aufweiten" der Hindernisse zur Extraktion begehbarer Pfade

Die Abbildung 5.5 illustriert dieses Vorgehen. Der Roboter mit seinen realen Abmaßen würde von der Pfadplanung ohne künstliches Vergrößeren der Hindernisse auf den Pfad 1 geschickt, wo er mit großer Wahrscheinlichkeit nicht zwischen den Hindernissen passieren könnte. Das Aufweiten der Hindernisse läßt damit den Pfad 1 gar nicht erst zu.
Das Maß der Aufweitung der Hindernisse hängt mit den Eigenschaften des eingesetzten Roboters eng zusammen, holonomen bzw. nicht-holonomen Antriebskonzepten.
Übliche Experimentalroboter, wie z.B. MauSI mit *differential drive* Antriebsmechanismus ist als nicht-holonomes System mit zwei mechanischen Freiheitsgraden, rotatorisch und translatorisch, anzusehen. Nach [8] ist ein System genau dann als nicht-holonom

anzusehen, wenn die Bewegungsabläufe seiner Aktoren untereinander abhängig (*path-dependent*) sind. Im Gegensatz dazu ist die holonome Bewegung eines Roboterarmes anzusehen, dessen Gelenke alle gleichzeitig ihre Bewegung aufnehmen können ohne daß dies Auswirkungen auf die Endstellung des Armes hat. Der nicht-holonome Roboter erreicht differierende Endpositionen in Abhängikeit davon, ob er sich zuerst dreht und dann fährt oder umgekehrt.
Übertragen auf die Pfadplanung heißt das, daß die Steuerung des Roboters die Pfadplanung eine holonome Kreisfläche als Bewegungselement vorgeben muß.

## 5.2 Prinzip des "virtuellen Zielpunktes"

Im Gegensatz zu den mehr oder minder vollständigen *Roadmap* Verfahren, ist bei der *Potential Field* Annäherung keine vollständige Pfadberechnung vor dem Start der Roboterbewegung notwendig. Auch die Kenntnis der genauen Lage aller Hindernisse wird nicht vorausgesetzt.
Benötigt wird ein Richtungsvektor auf den Zielpunkt des Pfades. Dieser Vektor wird entweder aus einer Zielbeobachtung abgeleitet oder über einen internen Beobachter mitgerechnet (gekoppelt). Die Zielbeobachtung kann auch durch indirekte Verfahren, z.B. GPS-Ortung o.ä., zur Selbstlokalisation ergänzt werden.
In [35] wird eine Robotersteuerung mit Hilfe von Fuzzy-Logik realisiert. Die Navigation des Roboters wird in diesem Fall von einem internen Beobachter kontrolliert. Aufgabe des Beobachters ist die hinreichend genaue Lageermittlung des Roboters im Arbeitsraum. Der Beobachter kreiert einen Zielvektor und führt damit den Roboter nach dem Prinzip der *Potential Field* Annäherung auf den Zielpunkt zu.
Dieses Steuersystem verfügt jedoch nicht mehr über einen Gesamtüberblick über den Arbeitsraum. Damit auf Hindernisse reagiert werden kann, muß der Zielpunkt beim Erkennen derselben verlegt werden.
Zur Zielpunktverlegung, d.h. zum Kreieren eines "virtuellen Zielpunktes" werden Regeln aufgestellt.

- Hindernisvermeidung und Verlangsamung vor Hindernissen
- Wandverfolgung
- Zielansteuerung

Mit Hilfe dieses Vorgehens kann der Rechenaufwand soweit gesenkt werden, daß die erforderlichen Algorithmen auf der mobilen Einheit implementiert werden können.

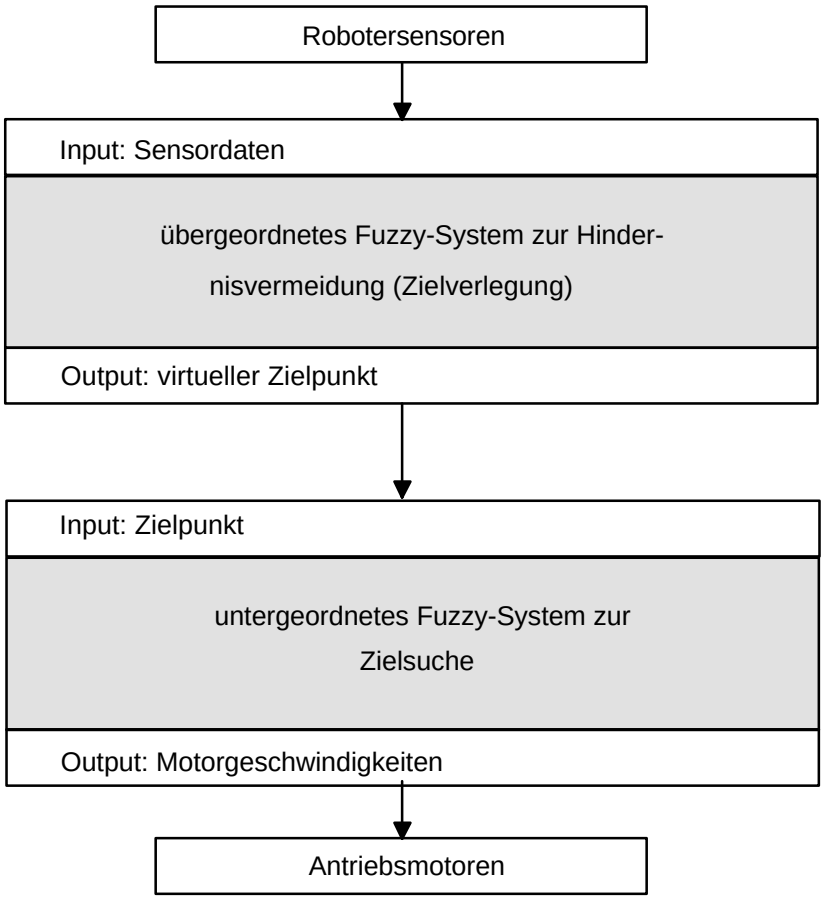



Bild 5.6 Steuersystem "virtueller Zielpunkt" aus [35]

Neben der Implementation auf dem Roboter "Khepera" ist diese Methode auch in einem Projekt [36] auf der Basis MathLab/Simulink in eine PC Plattform implementiert worden.
Ziel dieses Projekts ist die schnelle Überprüfung des Regelwerkes zur Hindernisvermeidung und Zielpunktgenerierung. Die eigentliche Berechnung des virtuellen Zielpunktes erfolgt nach den gleichen Regeln wie im Roboter. Zur besseren Bewertbarkeit ist an die Simulation eine Visiualisierung angeschlossen.
Mit Hilfe eines grafischen Editors wird der Arbeitsraum mit Hindernissen "gefüllt". Im Anschluß daran wird ein Roboter plaziert und der jeweils berechnete Zielpunkt dargestellt. Die Simulation erfolgt in Echtzeit, so daß der Betrachter beim Auftauchen von Hindernissen im Erfassungsbereich der Robotersensoren das Wandern des virtuellen

Zielpunktes auf dem Bildschirm verfolgen kann. Zur besseren Erkennung ist eine perspektive dreidimensione Bildschirmdarstellung möglich. Die Abbildung 5.7 zeigt einen Screenshot.

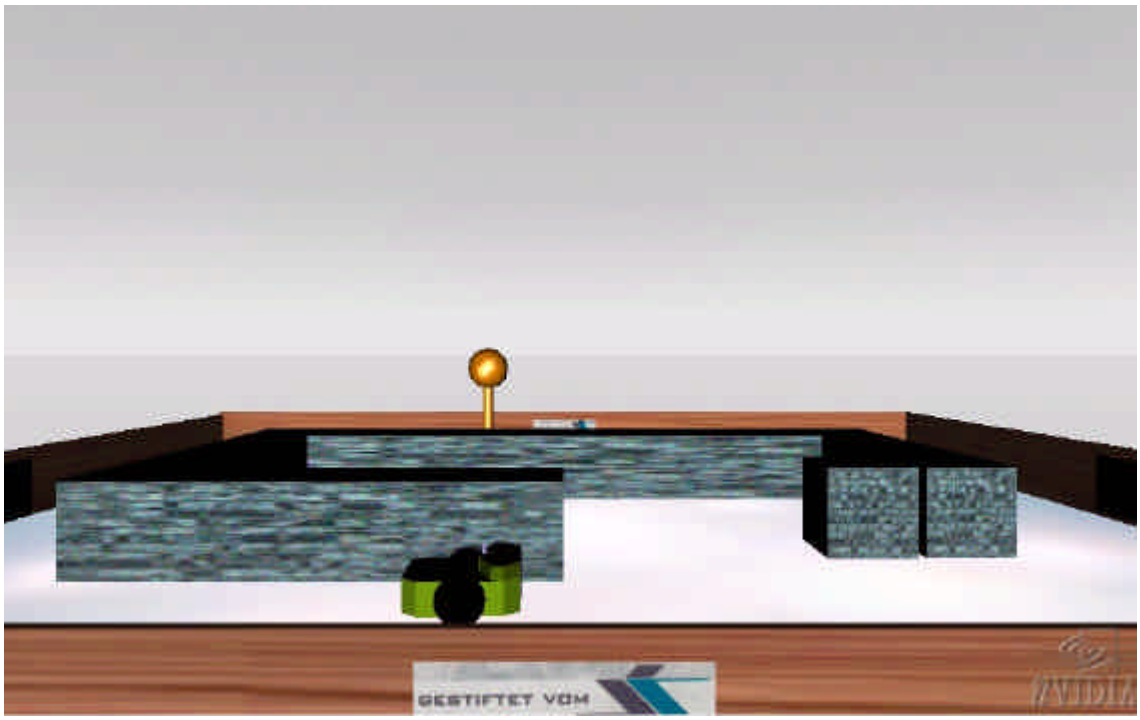



Bild 5.7 Screenshot der Simulationsumgebung [36]

## 5.3 Bahnführung durch rekursive Streckenzerlegung

Die bislang besprochenen Pfadplanungen legen keinen Wert auf die Kenntnis (*potential field method*) bzw. Speicherung (*roadmap*) der bereits absolvierten Wegstrecke. Warum diese Kenntnis durchaus von Nutzen sein kann, wird in späteren Abschnitten näher erläutert.

Ein Schwachpunkt des Prinzipes des virtuellen Zielpunktes ist die undefinierte Schrittweite der Berechnungszyklen. Die Berechnung des jeweiligen Zielpunktes muß so schnell sein, daß die Sensordaten rechtzeitig in die erforderlichen Motorgeschwindigkeiten umgesetzt werden.

Aus der Sicht einer Implementierung der Algorithmen auf einem Mikrocontroller sind natürlich Aussagen wie "rechtzeitig" oder "schnell genug" keinesfalls akzeptabel. Die Bewegungen des Roboters stellen harte Echtzeitbedingungen an die Programmlaufzeit.

Vom abgeschätzten relativ geringen Ressourcenbedarf erschien die Methode des virtuellen Zielpunktes so attraktiv, daß der Versuch unternommen wurde, diese Methode in zweierlei Hinsicht zu "diskretisieren".

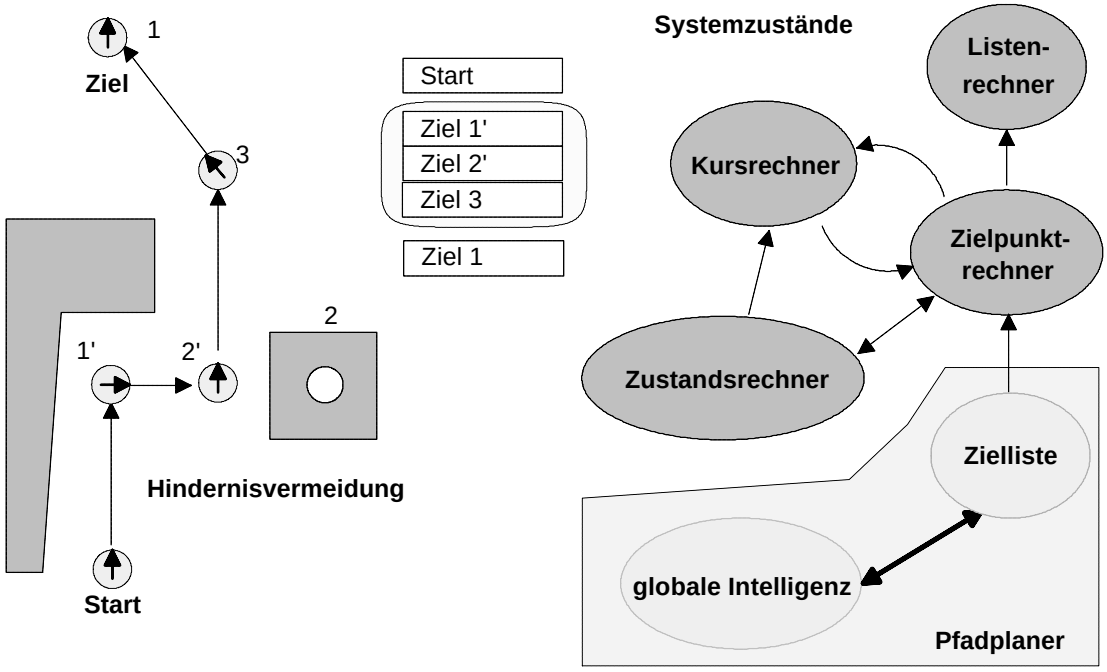



Bild 5.8 Zustandsmodell der Softwareimplementation im Roboter MauSI 1

Zur Implementation des Systems auf einem *low power* 8-Bit Mikrocontroller werden eine Reihe von Systemzuständen definiert, Zielpunktrechner (ZPR), Kursrechner (KR), Zustandsrechner (ZR), Listenrechner (LR).
Ein Pfadplaner, der bei der Erstimplementation aus Performancegründen in ein externes System (globale Intelligenz) verlagert wurde, ergänzt den Roboter.
Anhand des Beispieles aus der Abbildung 5.8 wird die Vorgehensweise erläutert. Der externe Pfadplaner zerlegt den zu absolvierenden Pfad in n Unterabschnitte. Der Wert für n richtet sich nach bestimmten Charateristiken der Gesamtroute.
Aus der Sicht des Roboters stellt sich der Gesamtweg als eine Liste von n Zielpunkten dar. Bei den ersten Experimenten mit dem Roboter MauSI 1 enthielt die übergebene Zielliste nur einen Zielpunkt. Der Zustandsrechner erkennt die Übergabe des neuen Zielpunktes und veranlaßt der Zielpunktrechner zur Ermittlung der notwendigen Steuerinformationen zum Erreichen des Zielpunktes durch den Roboter.
Jeder Zielkurs besteht aus Drehwinkel und Wegstrecke, d.h. die Approximation eines beliebigen Pfades im Arbeitsraum erfolgt durch Interpolation mit Hilfe von Streckenelementen.

Die ermittelten Daten übergibt der Zielpunkrechner an den Listenrechner, der diese Daten verwaltet und an den Kursrechner.
Der Kursrechner wandelt Winkel und Distanz in Steuerdaten für die Antriebsmotoren um. Da es sich beim MauSI 1 um eine nicht-holonome Roboterkonstruktion handelt, wird das Absolvieren jedes Streckenelementes in eine rotatorische und eine translatorische Komponente aufgeteilt. Die Reihenfolge der Ausführung dieser Komponenten lautet immer: zuerst Rotation dann Translation. Durch den Antrieb mittels *differential drive* und den achteckigen Aufbau des Roboters (Annäherung an Kreisfläche), wird sichergestellt, daß die rotatorische Wegkomponente immer ausgeführt werden kann und dabei keine zusätzliche Überwachung durch den Zustandsrechner erfordert.
Im Beispiel der Abbildung 5.8 steht der Roboter genau in Zielrichtung (Pfeilspitze), der Kursrechner würde keine rotatorische Komponente erhalten. Während der Roboter auf seinem Wegabschnitt auf den Zielpunkt zusteuert, überwacht der Zustandsrechner die Sensoren.
Am Wegpunkt 1' erkennt der Zustandsrechner ein Hindernis. Daraufhin wird der Kursrechner deaktiviert und der Zielpunktrechner gestartet. Aus den Sensorinformationen erhält der Zielpunktrechner einen neuen Zielpunkt (virtueller Zielpunkt) und den bereits erreichten Wegpunkt 1' als Koordinateninformation zugewiesen. Der Listenrechner muß den Wegpunkt 1' in seine Zielpunktliste aufnehmen.
Im Ergebnis der ZPR-Berechnungen bekommt der Kursrechner den Wegpunkt 2 als neuen Zielpunkt zugewiesen.
Im Beispiel kann der Roboter diesen Zielpunkt ebenfalls nicht erreichen, aber nach der Berechnung eines weiteren virtuellen Zielpunktes 3 (der dann erreicht wird) kann dann der vorgegebene Endpunkt der Wegstrecke angesteuert werden.
Dieses recht ausführlich beschriebene Szenario klingt komplex, ist aber an die Möglichkeiten des eingesetzen Mikrocontrollers sehr gut anpaßbar. Die verschiedenen Rechner, ZPR, ZR, KR und LR sind als Einzeltasks in einem Multitasking-System implementierbar. Infolge der strengen Limitierung der Hardwareresourcen dürfen nicht mehr als zwei Tasks gleichzeitig laufen. Der Zustandsrechner ist immer aktiv, er steuert das Aufsetzen bzw. Terminieren der anderen Tasks.
Die Ähnlichkeiten zum System in [35] sind offensichtlich. Die Hauptunterschiede bestehen in der Speicherung der wirklich abgefahrenen Wegpunkte und in der erheblichen Reduktion des Rechenaufwandes. Im Rahmen einer Präsentation anläßlich des 45. IWK in Ilmenau konnte *"MauSI 1"* erstmals präsentiert werden [20].

# 6 Verhaltensobjekte und Suchalgorithmen

Die Experimente in [20] deuteten einen erfolgversprechenden Entwicklungsweg an. Im Ergebnis weiterer Untersuchungen zeigte es sich, daß zwei wichtige Anforderungen noch zu lösen sind.
Die bislang erwähnten Pfadplanungen bzw. Bahnsteuerungen sind auf Einzelroboter zugeschnitten. Eine Pfadplanung für mehrere Systeme gelangt bei der Verwendung der bereits geschilderten Verfahren sehr schnell an die Grenze der Berechenbarkeit. Durch die Einführung orts- und zeitvarianter Hindernisse, das sind aus der Sicht eines Roboter die (festen) Hindernisse und die anderen Roboter (variable Hindernisse), steigt die Anzahl der erforderlichen Pfadneuplanungen schnell. Der Rechenaufwand nimmt weiter zu, wenn außerdem das Zeitintervall zwischen den Pfadplanungen kleiner wird.
Unabhängig von der Anzahl der eingesetzten Roboter wird entweder ein kompletter Pfad aus der Kenntnis der Kartendaten generiert, bzw. der Pfad während der Bewegung des Roboters berechnet.
In beiden Fällen ist der bereits zurückgelegte Weg nicht mehr von Interesse. Agieren jedoch mehrere Roboter gleichzeitig im Arbeitsraum spielen die bereits absolvierten Pfadabschnitte durchaus eine Rolle. Eine starke Reduktion des Rechenaufwandes ergibt sich nämlich dann, wenn nur eine Route ermittelt werden braucht.
Die Frage der Pfadplanung soll sich also auch auf die Führung mehrerer Roboter beziehen können.
Das andere Problem bezieht sich auf die latente Gefahr, mit dem vorhandenem Regelwerk ein virtuelles Ziel zu definieren, welches zu einer Blockade des Roboters führt (*dead lock* durch lokalen Extremwert). Die besondere Problematik dieses Zustandes liegt zum Teil auch darin, daß dieses Verhalten in der Implementation in [35] überhaupt nicht vom System erkannt wird.
Es ist also zwingend notwendig, Kausalitätskriterien für den Zustandsrechner (ZR) zu definieren, die es ihm gestatten, diese Zustände zu erkennen. Eine relativ einfache Variante besteht z.B. in der Abschätzung der Zeitdauer, die zum Erreichen des Zieles erforderlich wäre. Ein Überschreiten dieser Vorgabe wäre dann ein potentielles Alarmsignal. Eine andere Variante bewertet die Komplexität der rekursiv ermittelten Streckenabschnitte zur Hindernisvermeidung. Nach einer zu definierenden Anzahl der Rekursionen (Abbruchschranke) wird der Versuch den Zielpunkt auf der eingeschlagenen Route zu erreichen, abgebrochen. Im Rahmen der Genauigkeit des Kursrechners wäre dann sogar ein Rückfahren auf der eigenen Spur mit anschließender neuer Zielsuche mit modifiziertem Regelwerk zur Definition des virtuellen Zielpunktes möglich.

Noch besser wäre es, wenn es gelänge, alternative Pfade mit differierenden Regelwerken zu prüfen, um so eine optimale Hindernisvermeidung zu erreichen.
Die Abstraktion dieser Problematik führt zu folgender Aufgabenstellung:

- Definition eines Wegabschnitts mit Prognose des Regelwerkes zur Definition des virtuellen Zielpunktes. Diese Definition stützt sich auf vorhandene (globale) Daten, z.B. *Roadmap*. Dieses Element beinhaltet sowohl Daten als auch Vorschriften zum Handling dieser Daten. Im folgenden wird es als Verhaltensobjekt (*Behavior Object*) bezeichnet.

- Festlegen von unterschiedlichen Bewegungsmustern (*Moving Pattern*) zur Hindernisvermeidung. Ein solches Bewegungsmuster bezieht sich nur auf kleine räumliche Abschnitte, nach Möglichkeit in der Reichweite der Hindernisdetektoren (bei MauSI 2 liegt dies in der Größe einiger Zentimeter)

- Algorithmen zur formalisierten Bewertung der Bewegungsmuster über ein Segment des vorgegebenen Pfadabschnitts. Auch hier wird der Bewertungsbereich durch die Reichweite der Abstandsdetektoren eingegrenzt.

Die definierten Verhaltensobjekte bzw. Bewegungsmuster sollen zwar auf die Verwendung im Roboter "MauSI 2" optimiert werden, jedoch allgemeingültigen Charakter besitzen.

## 6.1 Aufbau des Verhaltensobjekts

Fortgeschrittene Hochsprachen, wie C++ oder Borland-Delphi, versuchen zunehmend, die immer komplexeren Anforderungen großer Softwareprojekte durch Modularisierung und Kapselung für den Programmierer zu vereinfachen. Besonders interessant ist in dieser Hinsicht die objektorientierte Programmierung (OOP). Wesentliches Kennzeichen dieses Vorgehens ist das Programmierobjekt.
Aus der Sicht der Programmierer wird das *Objekt* durch *Attribute* (Variablen) und *Methoden* (Prozeduren) beschrieben. Durch *Messages* (Botschaften) kommunizieren die Objekte untereinander.
Die Spezifizierung des Objektes in Methoden und Attribute ist ein weitgehender Schritt zur Formalisierung des Softwareentwurfes. Objekte lassen sich auch zu Klassen von Objekten zusammenfassen. Wird dieser Ansatz weiter formalisiert, kann ein Objekt als Basis zur Generierung weiterer ähnlicher Objekte dienen. Dieser Mechanismus, als Vererbung bezeichnet, ermöglicht die Ableitung neuer Objekte aus bereits vorhandenen. Ein Basisobjekt dient dabei als eine Art *Template* (Schablone), so daß die neu erzeugten Objekte die Eigenschaften des *Templates* erhalten.

Als beliebtes Beispiel zum Verdeutlichen der Eigenschaften eines Objekte wird gern ein Automobil herangezogen.
Die Attribute des Objektes *Auto* sind z.B. Zahl der Insassen oder Geschwindigkeit, Methoden wären z.B. bremsen bzw. beschleunigen.
Die Methoden eines Objektes können in erster Linie nur auf die Attribute des selben Objektes angewandt werden. Es findet deshalb eine strenge Kapselung von Daten und Funktionen (Prozeduren), die mit diesen Daten arbeiten können, statt.
Bei solch einfachen Objekten sind die Vorteile in der Anwendung nicht sofort sichtbar. Gelegentlich scheint der unvermeidbare Overhead dieses Programmieransatzes sogar wiedersinning und kontraproduktiv.

### 6.1.1 Verhaltensobjekt für "MauSI 1"

Für die Steuerung des Roboters "MauSI 1" wird in Anlehnung an ein solches Programmierobjekt ein Verhaltensobjekt kreiert. Die Methoden des Verhaltensobjektes sind im weitesten Sinne die Bewegungs- bzw. Verhaltensmöglichkeiten (Funktionen) des Roboters. Die Attribute sind als Parameter dieser Funktionen anzusehen.
Zur Steuerung des Roboters MauSI 1 wurde ein Objekt mit folgendem Aufbau genutzt:

```
/* Typdefinition Verhaltensobjekt "MauSI 1" */

typedef struct stObject /* Struktur für Verhaltensobjekt  (32 Byte groß)   */
     {
     byte bDirection;   /* Offset des aktuellen Drehwinkels zur Referenz   */
     byte bSpeed;       /* Geschwindigkeit                                 */
     int16 iXPosit;     /* Koordinaten des Startpunktes des Wegabschnittes */
     int16 iYPosit;
     int16 iXFinish;    /* Zielpunktkoordinaten                            */
     int16 iYFinish;
     byte bActivTask;   /* Funktions-Task (Methode)                        */
     byte bParameter;   /* Objektparameter (Attribute)                     */
     byte abArray[3];   /* Feld mit zusätzlichen Attributen                */
     word wMissionTime; /* Systemdaten: verstrichene Zeit                  */
     byte bMainPower;   /*              verbleibende Akku-Kapazität        */
     byte abAnalog[6];  /*              absolute Werte der IR-Sensoren     */
     byte abIrDist[8];  /*              normierte Werte der IR-Sensoren    */
     }stObject;
```

Das Objekt beinhaltet alle wesentlichen Information zur Kontrolle des Roboters innerhalb des spezifizierten Wegabschnittes. Der Zielpunktrechner (ZPR) benötigt Startpunkt, Zielpunkt und Offset des momentanen Drehwinkels zu einem Referenzpunkt, um daraus die nötigen Steuerdaten für die Antriebsmotoren zu ermitteln.
Eine Reihe von Daten, Akkukapazität, Werte der IR-Sensoren, müssen vom Betriebssystem des Roboters bereitgestellt werden. Die Roboterfunktionalität (Methode) verbirgt sich hinter dem Index auf den Funktionstask.

Die begrenzten Ressourcen von "MauSI 1" erlauben nur Ausführung von zwei unabhängigen Tasks. Da der Systemtask zur Ansteuerung der Peripheriemodule (Funk, Motoren, usw.) sowie zur Aktualisierung der Systemdaten (Systemzeit, Sensoren, etc.) immer aktiv sein muß, ist der Funktionstask eine Kombination aus Zustandsrechner (ZR) und eigentlicher Objektmethode. Der Übergang zwischen den Objekten erfolgt nach bestimmten (vordefinierten) Übergangsregeln.

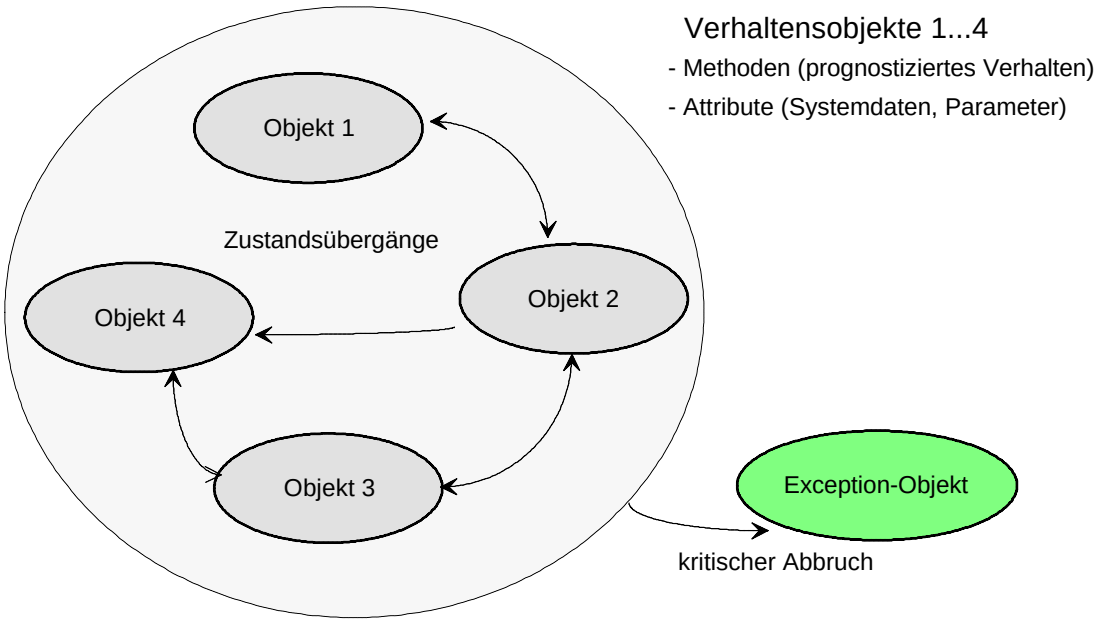



Bild 6.1 Implementation der Verhaltensobjekte beim "MauSI 1"

Die Versuche am Roboter "MauSI 1" mit dem beschriebenen Verhaltensobjekt verliefen anfangs vielversprechend. Die globale Intelligenz zerlegt den Gesamtpfad in eine Liste von Pfadabschnitten, prognostiziert nach bestimmten Regeln ein wegspezifisches Roboterverhalten und erzeugt daraus eine Liste von Verhaltensobjekten. Diese Liste geht per Funk an die Roboter. In [20] wurden erfolgreich Roboter von einer zentralen (globalen) Intelligenz durch ein Hindernisparcour geführt.

### 6.1.2 Verhaltensobjekt für "MauSI 2"

Mit zunehmender Komplexität des Arbeitsraumes steigt die Anzahl der Übergangsregeln zwischen den Verhaltensobjekten. Der einzige Ausweg, diesen Anstieg zu bremsen, bestand in der Erhöhung der Funktionalität des jeweils aktiven Verhaltensobjektes, d.h. zunehmende Anzahl von Methoden. Dieses Vorgehen widersprach dem Ansatz zur Modularisierung der Verhaltensobjekte und überstieg die verfügbaren Ressourcen des

Steuercontrollers rasch. Gleichzeitig bot das starre Korsett der Zustandsübergänge als einziges Verfahren zum Objektwechsel, nicht genügend Flexibilität, um auf lokale Ereignisse, die zum Zeitpunkt der Pfadplanung unbekannt waren, angemessen zu reagieren.
Der Roboter wurde deswegen mit einem leistungsfähigeren Controller ausgestattet und erhielt bei grundsätzlich gleicher Steuerung (über das Verhaltensobjekt) ein neu strukturiertes Verhaltensobjekt, welches die Möglichkeiten objektorientierter Programmierung besser ausnutzt. Insbesondere wurde auf die Implementierung von Botschaften (Callback-Funktionen) zur Interobjektkommunikation Wert gelegt.

```
/* Typdefinition Verhaltensobjekt "MauSI 2" */

typedef struct stObject          /* Struktur für Verhaltensobjekt          */
     {
     Methode fMethodeList[];     /* Referenz auf Methodenliste             */
     CallBackFunc fFuncList[];   /* verfügbare Callbackfunktionen          */
     bool (* fStepGen)(stMap *, stPoint, stPoint, stBehavior);
     byte bMethodeListIndex[];   /* Indizies auf Methoden                  */
     byte bFuncListIndex[];      /* Indizies auf Callbackfunktionen        */
     stPoint (* fStepGen)(stMap *, stPoint, stBehavior); /* Stepgenerator  */
     stBehavior stBehObj;        /* prognostiziertes Verhalten             */
     stPoint stPosition;         /* aktuelle Position                      */
     stPoint stStart;            /* Koordinaten des Startpunktes           */
     stPoint stVirtuell;         /* virtueller Zielpunkt                   */
     stPoint stZiel;             /* Zielpunktkoordinaten                   */
     stSysData stData;           /* Systemdaten: verstrichene Zeit         */
                                 /*              Akku-Kapazität            */
                                 /*              Werte der IR-Sensoren     */
     stMap stLocalMap;           /* Kartendaten des Wegstreckenelements    */
     }stObject;
```

Dieses Verhaltensobjekt ist in seiner Funktionalität gegenüber dem des Roboters "MauSI 1" wesentlich ergänzt und erweitert worden. Neu sind die Funktionspointer auf die Methoden (*fMethodeList[]*) des Objektes. Diese Pointer verweisen auf allgemein verfügbare Prozeduren (Funktionen), die allen Objekten zur Verfügung stehen (beim Multitasking-Betriebssystem müssen diese Funktionen reentrant sein).
In der Liste der verfügbaren Callbackfunktionen sind die Schnittstellen zwischen Objekten bzw. zur Systemtask aufgeführt. Die Systemtask hat auch hier die Aufgabe bestimmte Attribute des Objektes (Zeit, Sensorwerte, etc.) laufend zu aktualisieren. Auch die laufende Mitrechnung des absolvierten Weges (Kursrechner) wird außerhalb des Objektes realisiert.
Gänzlich neu ist die Methode *fStepGen(stMap*, stPoint, stPoint, stBehavior).* Mit dieser Funktionalität kann der Roboter Ausweichpfade berechnen, die hauptsächlich auf lokal gewonnenen Daten und eines von der globalen Intelligenz geschätzten Verhaltens basieren. Dieser "Zuggenerator" befähigt die lokale Intelligenz (Roboter) zu eigenen Entscheidungen.

## 6.2 Laborexperimente mit der Plattform MauSI 2

### 6.2.1 Abbildung des Arbeitsraumes durch IR-Sensorscan

Im Abschnitt 3.3.4.1 ist die Ausstattung des Roboters mit IR-Sensoren beschrieben worden.
Trotz der Anstrengungen, die in [17] unternommen wurden um die Meßwerte zu exakteren Entfernungsangaben zu verrechnen, bleibt das Ergebnis relativ ungenau. Es sind lediglich Entfernungsschätzungen möglich. Um für den Praxisversuch reproduzierbare Werte zu gewinnen, wird der Arbeitsraum nach den Besonderheiten der Sensorik gestaltet.
Die Grundfläche wird mit schwarzem Papier bespannt und alle Hindernisse werden aus einheitlich hellgrauer diffus reflektierender Pappe hergestellt. Das folgende Diagramm zeigt den Zusammenhang zwischen Meßwert und Objektentfernung.

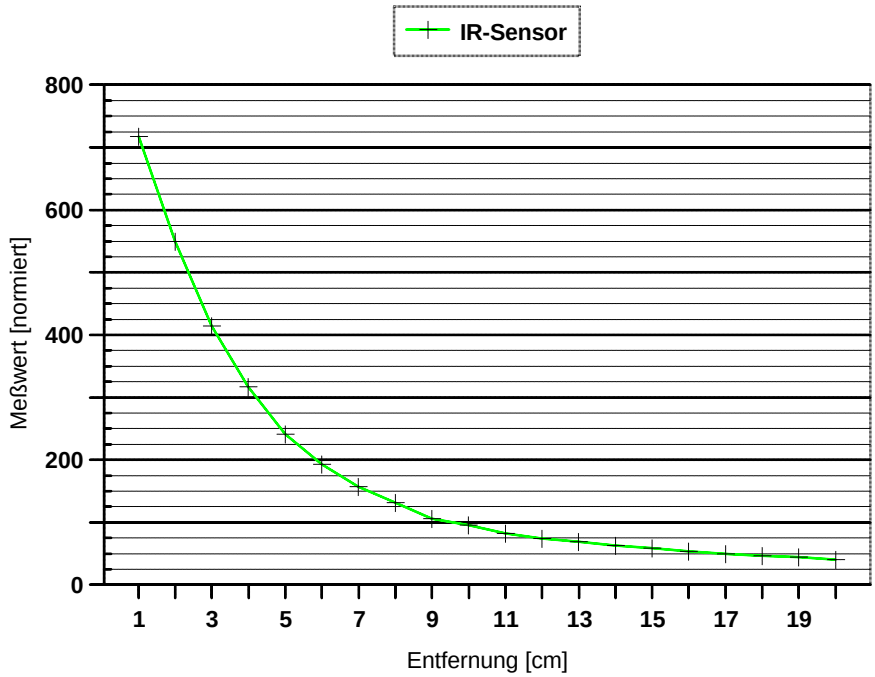



Bild 6.2 Zusammenhang zwischen IR-Sensorwerten und Objektentfernung

Infolge der stark nichtlinearen Kennlinie werden Meßentfernungen über 10 cm Objektabstand nur noch sehr ungenau aufgelöst, auf einen Erfassungsbereich größer 10 cm wird deswegen verzichtet.
Der Roboter ist mit insgesamt vier IR-Abstandsdetektoren ausgerüstet. Drei Sensoren sind mit "Blick" in Fahrtrichtung, einer mit "Blick" nach hinten angeordnet. Die als

IR-Sender fungierenden IR-LEDs tragen je einen Tubus, der den Lichtkegel entsprechend fokussiert. Damit wird u.a. der Erfassungswinkel der Sensoren an die Sendeeigenschaften der LEDs angepaßt.

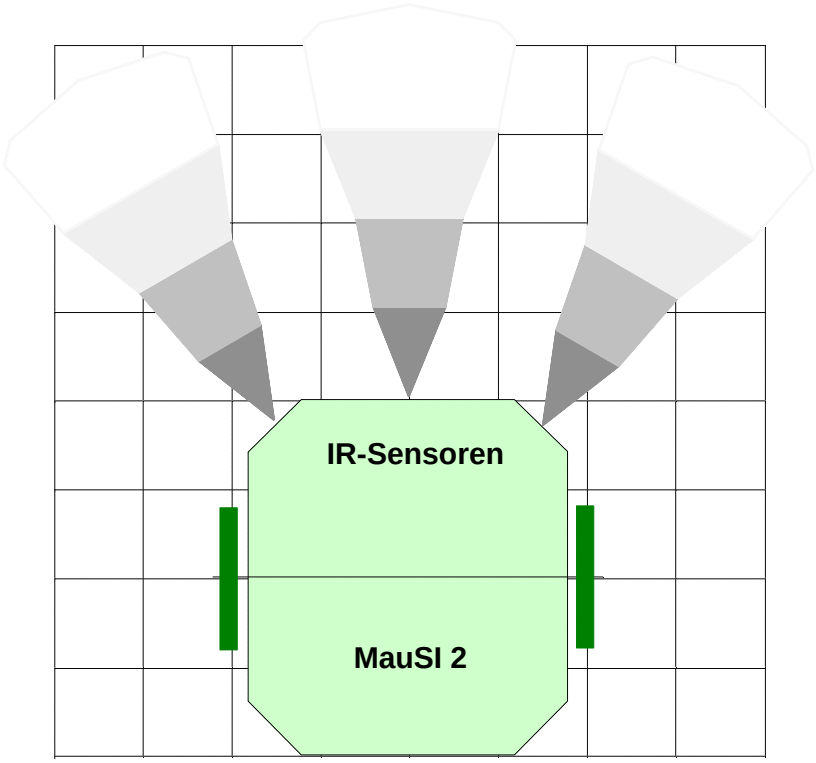



Bild 6.3 Erfassungsbereiche der IR-Sensoren im Frontbereich

In der Abbildung 6.3 ist der Zusammenhang zwischen Erfassungsbereich und Rasterung des Arbeitsraumes zu sehen. Die Sensoren können etwa einen Bereich von 8 Rasterelementen Breite (8 x 2,5 cm = 20 cm) überstreichen. Damit kann unter der Vorausetzung, daß Hindernisse Abmessungen von min. einem Rasterelement besitzen, eine Kollisionsüberwachung mit hinreichender Sicherheit erfolgen. Die unterschiedlichen Grauabstufungen der Scanbereiche deuten die Empfindlichkeit der Sensoren an.

### 6.2.2 Meßdatenaufnahme

Mit einer "Testfahrt" werden die Meßwerte der IR-Sensoren auf ihre Brauchbarkeit hin untersucht. Dazu wird ein Testprogramm in den Controller des Roboters geladen, welches die Sensorwerte über die serielle Schnittstelle ausgibt. Zur Aufnahme der Werte wird der Roboter auf einem Pfad im 2,5 cm Raster schrittweise bewegt. Die folgende Tabelle beinhaltet die gemessenen Werte.

| Schritt | Sensor Vorn | Sensor Links | Sensor Rechts | Bemerkung |
|---|---|---|---|---|
| Bild 01 | 56 | 28 | 33 | |
| Bild 02 | 93 | 30 | 39 | |
| Bild 03 | 174 | 32 | 43 | |
| Bild 04 | 488 | 32 | 48 | |
| Bild 05 | 716 | 36 | 51 | Abbruchbedingung erreicht |
| Bild 06 | | | | Drehung, keine Sensorauswertung |
| Bild 07 | 70 | 50 | 37 | |
| Bild 08 | 100 | 53 | 46 | |
| Bild 09 | 179 | 51 | 70 | |
| Bild 10 | 494 | 54 | 142 | Wendepunkt erreicht (kein Abbruch) |
| Bild 11 | | | | Drehung, keine Sensorauswertung |
| Bild 12 | 41 | 82 | 81 | |
| Bild 13 | 38 | 85 | 81 | |
| Bild 14 | 41 | 95 | 76 | |
| Bild 15 | 40 | 100 | 71 | |
| Bild 16 | 37 | 99 | 69 | |
| Bild 17 | 30 | 98 | 31 | |
| Bild 18 | 31 | 89 | 20 | |
| Bild 19 | 32 | 103 | 20 | |
| Bild 20 | 31 | 104 | 19 | |
| Bild 21 | 31 | 87 | 17 | |
| Bild 22 | 24 | 38 | 15 | |
| Bild 23 | 27 | 30 | 19 | Wendepunkt erreicht (kein Abbruch) |
| Bild 24 | | | | Drehung, keine Sensorauswertung |
| Bild 25 | 52 | 25 | 114 | |
| Bild 26 | 56 | 27 | 96 | |
| Bild 27 | 60 | 21 | 91 | |
| Bild 28 | 65 | 30 | 61 | Wendepunkt erreicht (kein Abbruch) |
| Bild 29 | | | | Drehung, keine Sensorauswertung |
| Bild 30 | 24 | 24 | 27 | |
| Bild 31 | 24 | 23 | 24 | |
| Bild 32 | 27 | 24 | 17 | Zielpunkt erreicht |

Bild 6.4 Meßwerte aus dem Arbeitsraum

Die Entscheidung ob der eingeschlagene Kurs beibehalten werden kann oder abgebrochen werden muß stützt sich auf die im Bild 6.4 aufgeführten Sensordaten.
Zur besseren Bewertung und Datenzuordnung sind in der Abbildung 6.5 die Meßstationen in einer Abfolge von Einzelaufnahmen des abgefahrenen Kurses dargestellt.

Die Position "Bild 05" in Abbildung 6.4 ist farblich hervorgehoben. An dieser Stelle signalisiert der Zustandsrechner (ZR) eine Abbruchbedingung. Der IR-Sensor detektiert das Unterschreiten eines Mindestabstandes und generiert eine Kollisionswarnung.

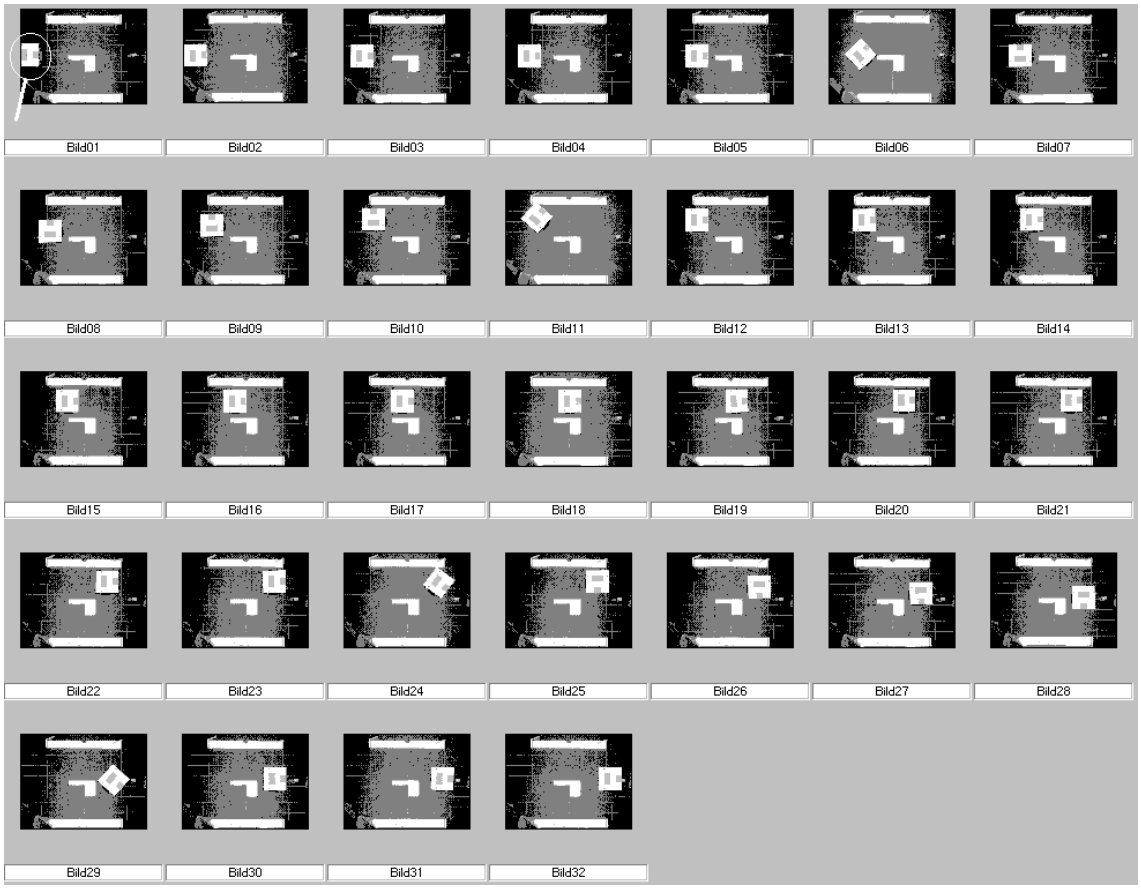



Bild 6.5 Bildsequenz der absolvierten Meßstrecke des Roboters

Aus den Daten des Sensors kann nicht unmittelbar auf einen Ausweichkurs geschlossen werden. Lediglich der übergeordnete Beobachter erkennt, daß das Hindernis nur auf der linken (oberen) Seite umfahren werden kann. Wird der andere Kurs gewählt, kann der Zielpunkt nicht erreicht werden.

Infolge meßtechnischer Ungenauigkeiten, z.B. ungenaues der Pfadplanung zugrunde liegendes Kartenmaterial, kann die globale Intelligenz jedoch nicht gleich den richtigen Kurs errechnen. Bei der Bildaufnahme für die Abbildung 6.5 sind bewußt sehr niedrig aufgelöste Fotos gemacht worden. Die niedrige Auflösung ist als Maß der Kartenungenauigkeit zu sehen. Weiterhin könnte das Hindernis in der Mitte des geplanten Kurses auch eine Bildstörung darstellen. Das Einbeziehen aller Störungen könnte eine erfolgreiche Pfadplanung unter Umständen verhindern. Es scheint sinnvoller zu sein, an dieser Stelle lediglich potentielle Ausweichstrategien zu prognostizieren und der lokalen

Intelligenz als Vorschlag bei der Generierung des Verhaltensobjektes für diesen Wegabschnitt zu übergeben.

### 6.2.3 Kursmitrechnung und dynamischer Kartenabgleich

Aus diesen Überlegungen zeichnet sich die Notwendigkeit einer stückweisen Pfadberechnung (Zielpunktgenerierung) auf der Seite der lokalen Intelligenz ab. Für die Praxis bedeutet das, daß der Roboter Kenntnis über die Umgebung seines vorgegebenen Kurses erhalten muß.
Zum Startzeitpunkt können dem Roboter nur die Informationen, die auch die globale Intelligenz besitzt, zugänglich gemacht werden. Diese Informationen stellen einen Teil des Datenanteils des Verhaltensobjektes (Attribut: *stLocalMap*).
Die eigentliche Kursmitrechnung ist Teil der Funktionalität der Systemtask des Roboters. In zyklischen Abständen wird dafür das Attribut *stPosition* des jeweils aktiven Objektes aktualisiert. Der Kursrechner stützt sich dazu auf die Daten der Radencoder und die Werte des Beschleunigungssensors.
Aufgabe des aktiven Objektes ist nun diese Wegstreckenänderungen mit den Sensorwerten der IR-Distanzschätzern zu verknüpfen und mit den lokal verfügbaren Karteninformation abzugleichen. Wegen der geringen Präzision der IR-Meßwerte, werden nur die Kartendaten aus der unmittelbaren Umgebung des Roboters auf ihre Relevanz geprüft. Es scheint derzeit nicht möglich, mit Hilfe der vorgegebenen Kartendaten und der lokal gewonnenen Meßwerte eine Selbstlokalisation des Roboters durchzuführen.

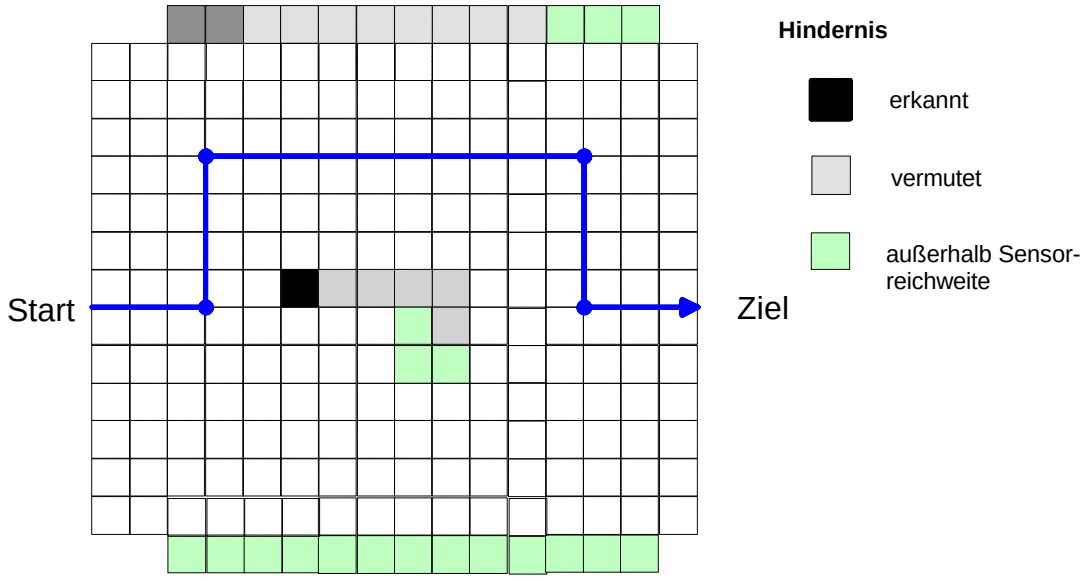



Bild 6.6 interne Karte des Beispielkurses von Bild 6.5

Die Abbildung 6.6 zeigt die vom Roboter lokalisierten Hindernisdaten. Aus der Sicht der Robotersensorik ist lediglich ein Feld (schwarz markiert) als Hindernis erkannt worden. Die grauen Bereiche liegen zwar in der Detektionsentfernung der IR-Sensoren, ihre genaue Anordnung (wie im Bild dargestellt) ist jedoch in der Realität so nicht möglich. Für den Roboter sind die grauen Bereiche zwar potentielle Hindernisse, da aber die Kursabbruckkriterien nicht erfüllt sind, werden diese Felder ignoriert.
Der Roboter navigiert durch seinen internen Kursrechner kontrolliert auf einem rechtwinkligen Koordinatennetz durch den Arbeitsraum. Die lokale Bahnplanung erlaubt nur Drehungen auf der Stelle, Rotation in Vielfachen von 90 Grad bzw. geradlinige Bewegungen, Translationen in Vielfachen von 25 mm (50 Steps der Radencoder).
Da sich die Hindernisse (auch unter Modellbedingungen) kaum an dieses Raster halten, wird nach der Erfüllung einer vollständigen Abbruchbedingung (Hindernis erkannt) auf den letzten erreichten Gitterpunkt zurückgefahren.
Eine weitere Vereinfachung in der Kursberechnung auf der lokalen Seite, besteht im Verzicht auf Koordinatentransformationen der Kartendaten. Eine globale Pfadplanungen ausschließlich mit 90 Grad Wendungen zu planen, scheint wenig effektiv. Die globale Pfadplanung ist deswegen in der Wahl der Drehwinkel nur durch die minimal erreichbaren Drehungen des Roboters, bei "MauSI 2" ca. 11,25 Grad, eingeschränkt.

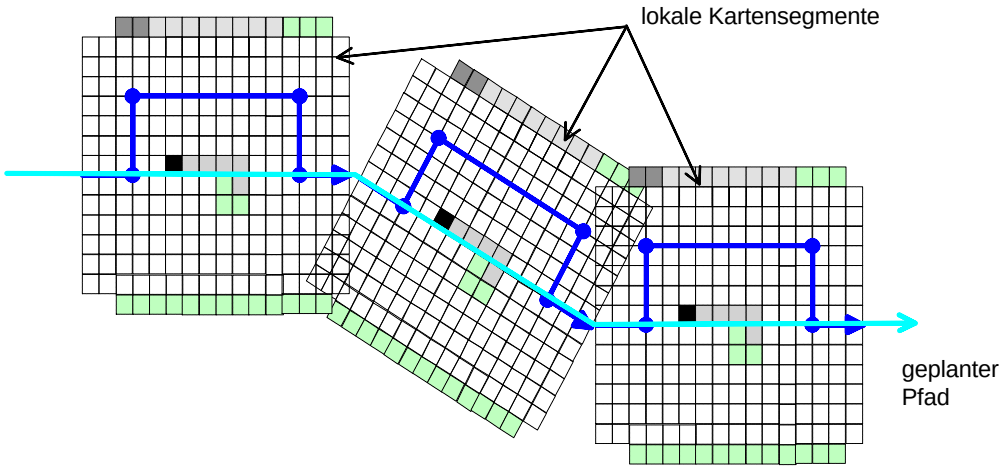



Bild 6.7 stückweise Pfadplanung mit wahlfreien Drehwinkeln zwischen den Teilstücken

Für jedes Teilstück des geplanten Pfades werden die dazugehörigen Kartendaten um den Drehwinkel des Roboters beim Einsteuerung in das Wegelement rotiert. Diese Rechenoperationen erfordern Speicherplatz und Rechenleistung, die auf dem Roboter nicht bereitstehen. Die Abbildung 6.7 verdeutlicht das Verfahren.

## 6.3 Symbolische Suchstrategien

Aus den vorausgegangenen Diskussionen und den besprochenen Beispielen entsteht die Forderung nach einem gewissen Maß an Entscheidungsfreiheit auf der lokalen Ebene.
Die Untersuchungen zur Thematik des virtuellen Zielpunktes ergaben, daß unter Zugrundelegung eines Regelwerkes eine intelligente Kollisionsvermeidung und Zielneuplanung möglich ist. Als weiteres Ergebnis zeigte sich jedoch auch, daß sich kein Regelwerk finden ließ, welches in jedem Fall neben der Kollisionsvermeidung eine widerspruchsfreie Zielneuplanung erlaubte.
Das Verhaltensobjekt stellt dem Roboter eine Prognose zum, aus der Sicht der globalen Intelligenz, erfolgreichsten Vorgehen der Zielneuplanung. Damit ist der erste Schritt zur Vermeidung lokaler Extremwerte im *Potential Field* bei der Ermittlung neuer virtueller Zielpunkte gegeben.
Dieses Verfahren funktioniert solange einwandfrei, wie keine signifikanten Abweichungen des Roboters vom vorgegebenen (idealen) Pfad bis zum prognostizierten Hindernis auftreten.

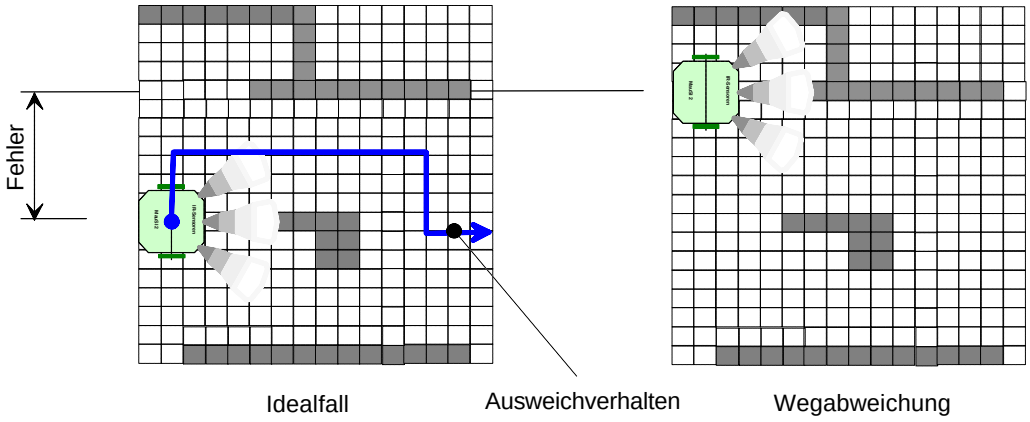



Bild 6.8 ideale bzw. fehlerbehaftete Annäherung an ein Hindernis

In realitätsnahen Szenarien ist jedoch mit einer Abweichung der Roboterbewegung von der Ideallinie infolge von Störungen zu rechnen.
Die Abbildung 6.8 demonstriert einen derartigen Fall. Der Roboter steuert in der rechten Bilddarstellung mit einer Abweichung vom Idealweg in das Pfadsegment ein. An der im Bild eingetragenen Stellung würde der Zustandsrechner eine Abbruchbedingung feststellen. Eine kritiklose Ausführung des prognostizierten Ausweichverhaltens würde im Beispielszenario praktisch sofort eine erneute Abbruchbedingung generieren.

Damit wird die Problematik der Formulierung widerspruchsfreier allgemeingültiger Regelwerke zur Kollisionsvermeidung deutlich.

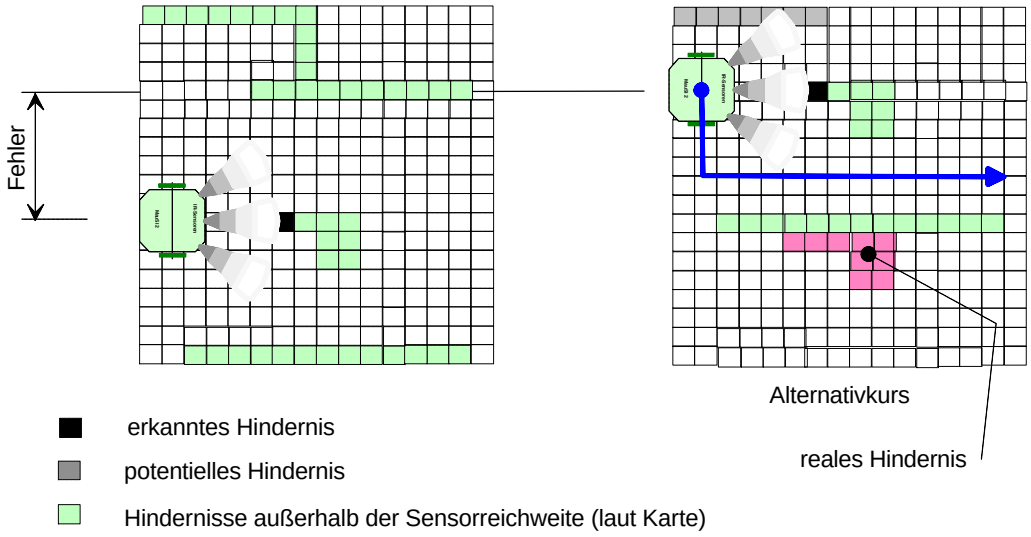



Bild 6.9 Errechnung eines Alternativkurses durch den Roboter

Der Roboter muß das vorgeschlagene Prognoseverhalten verwerfen und eine Alternativlösung suchen. Dazu müssen die aktuell gewonnenen *Roadmap* Informationen mit den Vorgabewerten abgeglichen werden.

Das Bild 6.9 zeigt in der rechten Bildhälfte die Kartendarstellung aus der Sicht des Roboters. Da diesem die Wegabweichung unbekannt ist, hat er seine Vorgabekarte mit zusätzlichen potentiellen Hindernissen aktualisiert. Potentielle Hindernisse sind solche, die zwar im Erfassungsbereich der Sensoren auftauchen, jedoch keine Abbruchbedingung generieren.

Wird nun vom Zustandsrechner eine Abbruchbedingung generiert, würde die Auswertung der Kartendaten zum Verwerfen des Vorgabeausweichverhaltens führen, und der Roboter muß einen, den aktualisierten Kartendaten entsprechenden, neuen Kurs planen.

Das eingangs erst nur zur Minimierung des Rechenaufwandes eingeführte rechtwinklige Koordinatennetz zeigt nun einen zweiten entscheidenden Vorzug. Die möglichen Bewegungselemente des Roboters lassen sich auf wenige einfache Grundelemente reduzieren. Im Wesentlichen sind dies Abfolgen von Drehungen auf der Stelle um jeweils 90 Grad mit anschließender gradliniger Bewegung um n Rasterschritte. Diese starke Vereinfachung resultiert aus den begrenzten Rechenleistungen des Steuercontrollers des Roboters.

Es liegt also nahe, diese Bewegungselemente weiter zu formalisieren und daraus quasi Symbole zu generieren. Gleichzeitig erscheint es ebenso sinnvoll, auch die Kartendaten in

ähnlicher Weise zu symbolisieren. Denkbar sind hier Symbole für durch die Robotersensorik festerkannte Hindernisse, Hindernisse aus dem vorgegebenen Kartenmaterial sowie potentielle Hindernisse, die im Erfassungsbereich der IR-Sensoren liegen, aber keine Abbruchbedingung erzeugen.
Unter diesen Voraussetzungen wird der Roboter befähigt, sowohl den prognostizierten Ausweichkurs auf Brauchbarkeit zu untersuchen, als auch selbstständig Alternativrouten möglichst optimal auszusuchen.
Damit entstehen folgende Forderungen für die Softwareimplementierung:

- *Zugerzeugung:* Bevor überhaupt an Zugauswahl zu denken ist, muß ein Programm alle legalen – d.h. regelkonformen – Züge erzeugen können, die in der jeweiligen Stellung möglich sind (und somit zu neuen Stellungen führen). Diese Aufgabe übernimmt der *Zuggenerator* des Programms.

- *Stellungsbewertung:* Aus der Anzahl der möglichen zu bewegenden Elemente muß von der Software eine Auswahl des erfolgversprechendsten Zuges getroffen werden.

- *Baumsuche:* Die Generierung aller möglichen Züge einer gegebenen Ausgangsstellung führt zu einer Vielzahl von Varianten die sehr schnell zu komlexen Baumstrukturen anwachsen. Dieser Suchbaum muß vom Programm auf möglichst effiziente Weise durchforstet werden..

- *Ruhesuche:* Der Abbruch der Berechnungen bei einer fixen Suchtiefe erscheint ungünstig, die Berechnungen müssen so weit in die Tiefe geführt werden, bis entweder die Endstellung (Zielpunkt erreicht) oder ein anders "Ruhekriterium" zum erfolgreichen Abbruch der Baumsuche führt.

Diese Vorgaben sind denjenigen, die bei der Konzeption von Strategiespielen nötig sind, sehr ähnlich. Erstmals formuliert wurden sie von Claude Shannon im Zusammenhang mit der Erörterung von Computerschachprogrammen.

### 6.3.1 Stellungsbewertung und Zuggenerator

Die Ähnlichkeit symbolischer Kartendaten und formalisierter Bewegungszüge mit Spielstrategien aus dem Bereich des Computerschach ist frappierend. Im weiteren wird deswegen versucht, dort gefundene Lösungsansätze auf die Problematik der Pfadplanung des Roboters anzuwenden.
Ein wesentliches Problem beim Computerschach stellt die Stellungsbewertung dar. Der Computer muß anhand bestimmter Kategorien ermitteln, welche Figur den höchsten

Effekt im Sinne der Gewinnmaximierung bietet. Kriterien sind z.B. Gewinn oder Verlust von Figuren.
Die Bewertung nach "Materialwerten" der verschiedenen Figuren ist rechentechnisch am einfachsten, nimmt jedoch in keiner Weise auf die Qualität der Spielstellung, d.h. strategischer Planungen, Rücksicht. Die ersten Schachprogramme waren demzufolge anfällig gegenüber angeboteten "Opfern".
Eine Stellungsbewertung im eigentlichen Sinn ist beim Einsatz nur eines Roboters nicht notwendig, da Züge nur für diesen einen Fall generiert werden müssen. Sind jedoch mehrere Roboter in Formation unterwegs, ist es unter Umständen zweckmäßig, eine Stellungsbewertung durchzuführen um zu entscheiden welcher Roboter zuerst bewegt wird.
Damit reduziert sich die zu implementierende Software auf die Programmierung des Zuggenerators für einen Roboter.
Die Kartendaten werden in ein symbolisches Spielfeld konvertiert. Dabei stellt sich die Frage, in welcher Form und vor allem warum die Hindernisse "gegnerische" Spielfiguren symbolisieren sollen? Die Hindernisse werden ja wohl kaum "zurückschlagen".
Eines der Probleme realer Robotersystems ist die bereits genannte Grenze in der genauen Abbildung der Umgebung als exakte Karteninformation. Ein Beispielszenario erläutert die Problematik. Der Roboter fährt in den spezifizierten Arbeitsraum ein. Ein Kartenabbild dieses Raumes gehört zu den Informationen, die er von der globalen Intelligenz erhalten hat. Mit Hilfe seiner Sensoren verifiziert er beim Durchqueren des Arbeitsraumes diese Datenbasis. Durch unvermeidliche Meßfehler bei der Wegmitkopplung werden Hindernisse an Stellen vermutet, an denen sie nicht existieren bzw. werden vermeindliche Hindernisse in den Kartendaten ausradiert, da die Sensorik an falscher Stelle sucht.
Diese Effekte sind einer zunehmend ungenauer werdenden Datenbasis gleichzusetzen. Die Nachbildung dieser Unschärfe erfolgt durch eine geometrische Ausweitung der Hindernisse während der Durchquerung des Pfadabschnittes. Außer in der unmittelbaren Umgebung des Roboters (Sensorreichweite) werden nach jedem Berechnungsschritt die Hindernisse größer.
Das mag zuerst unlogisch erscheinen, reflektiert aber die real auftretenden Ungenauigkeiten des internen Kursrechners sehr gut in die lokale Pfadplanung. Sind beispielweise mehrere Hindernisse mit unterschiedlichen aber passierbaren Abständen zueinander vorhanden, wird die lokale Pfadplanung, wenn sie durch Fehler vom Idealweg abgekommen ist, mit großer Wahrscheinlichkeit den breiteren Weg wählen, da die engere Passage einen "Bedrohungswert" erhalten hat, der von der Durchfahrt abhält.

Innerhalb der Genauigkeit der Sensoren werden mit jeder Ausweichbewegung neue Meßdaten gewonnen, die die Kartendaten laufend aktualisieren.
Die Ausweichpfadplanung besitzt deswegen, unabhängig von den ständig (künstlich) ungenauer werdenden entfernten Hindernissen, ein genaues Abbild der unmittelbaren Umgebung.
Die Aktualisierung der Kartendaten basiert auf der internen Koordinatenmitrechnung. Durch Fehleraddition ist mit einer ständig wachsenden Differenz von wahrem und mitgekoppeltem Standort auszugehen. Dieser Fehler muß als Schätzwert vom Kursrechner ermittelt werden, z.B. indem mit dem Anfangswert 0 (am Startpunkt) jede Ausweichbewegung diesen Wert inkrementiert und ihn gegen einen Grenzwert laufen läßt. Das Überschreiten dieses Grenzwertes ist das finale Abbruchkriterium, nach dem der Roboter eine Neulokalisierung benötigt.
Zusammenfassend läßt sich formulieren, daß durch abschnittsweise Vorgabe von Kartendaten eine lokale Pfadneuplanung innerhalb kleiner Strecken (etwa 10-fache Sensorreichweite) zur Hindernisvermeidung möglich ist. Diese Pfadneuplanung ist mehrfach nacheinander im gleichen Wegabschnitt (mehrere Hindernisse oder falscher Ansatz wegen ungenauer Kartendaten) möglich. Ein Schätzwert zur Abweichung zwischen wahrem und gekoppeltem Standort dient als Abbruchbedingung nach erfolgloser Zielansteuerung.
Da eine Bewertung der Stellung bei nur einem Roboter nicht notwendig ist (es ist nur ein Symbol vorhanden, welches bewegt werden kann) erzeugt nun der Zuggenerator mögliche Bewegungen die den Roboter um das Hindernis führen. Die möglichen Züge sind in einer Liste (*MoveList*) abgelegt. Ein Suchalgorithmus wählt den besten Zug.

### 6.3.2 Der Minimax-Suchalgorithmus

Der Minimax-Algorithmus ist der ineffektivste, dafür aber der einfachste Algorithmus zur Zug-Auswahl. Der im Beispiel gezeigte Pseudo-Code stammt aus [39].
Er probiert einfach alle Möglichkeiten bis zu einer bestimmten Suchtiefe durch. Der Namensgebung Minimax liegt eine Besonderheit der Stellungsbewertung zugrunde: Schachprogramme bewerten die Stellung mit ganzen Zahlen. Die Bewertung mit Null steht für eine ausgeglichene Stellung. Positive Zahlen repräsentieren einen Stellungsvorteil von Weiß, negative Zahlen einen von Schwarz. D.h. um so größer der Wert der Stellungsbewertung, desto größer der Vorteil für Weiß und um so kleiner der Wert der Stellungsbewertung, desto größer der Vorteil von Schwarz. Weiß versucht die Stellungsbewertung zu maximieren, und Schwarz versucht sie zu minimieren.

Der Minimax-Algorithmus läßt sich auf einfache Weise rekursiv programmieren (siehe Pseudo-Code).

```
Konventionen: beim ersten Aufruf enthält Depth den Wert 1, MaxDepth wird >= 1
gewählt und Color enthält den Wert des Zugrechtes (White/Black).

Prozeduren: EvaluatePosition bewertet eine auf dem internen Brett befindliche
Stellung und GenerateMoves ermittelt die möglichen Züge. MoveForward und
MoveBack ziehen intern Züge vor bzw. zurück.

         Const White      =      1;
               Black      =     -1;
               MaxInteger =  32767;
               MinInteger = -32768;

         Function MiniMax (Color, Depth, MaxDepth : Integer) : Integer;
         var Value, MaxValue ,
                    MinValue : Integer;

         begin
           if Depth = MaxDepth then
              MiniMax := EvaluatePosition (Color)

           else begin
              MinValue = MaxInteger;
              MaxValue = MinInteger;
              GenerateMoves(Color, MoveList);
              For Each Move in MoveList do
              begin
                  MoveForward (Move);
                      Value := MiniMax (-Color, Depth +1, MaxDepth);
                      if Color = White then
                         if Value > MaxValue then MaxValue := Value;

                      if Color = Black then
                         if Value < MinValue then MinValue := Value;
                  MoveBack (Move);
              end;
              if Color = White then MiniMax := MaxValue
                               else MiniMax := MinValue;
           end;
         end;
```

Zuerst wird überprüft, ob die maximale Rechentiefe erreicht ist. Ist dies der Fall, so wird die vorliegende Endposition bewertet. Ist die maximale Rechentiefe noch nicht erreicht, so werden MinValue und MaxValue zunächst auf die denkbar schlechtesten Werte gesetzt (MinValue für Schwarz groß, MaxValue für Weiß klein).
Nach der Initialisierung werden die möglichen Züge mit GenerateMoves ermittelt und in der nachfolgenden Schleife durchprobiert. Durch den rekursiven Aufruf werden automatisch alle Varianten nach einem probierten Zug untersucht. Das Ergebnis wird in Value festgehalten. Werden in den nachfolgenden Verzweigungen der aktuellen Rekursionsebene weiße Züge untersucht (Color=White), so wird maximiert, sonst minimiert. Die bisher beste Bewertung wird in MaxValue oder MinValue gespeichert und am Ende zurückgegeben.

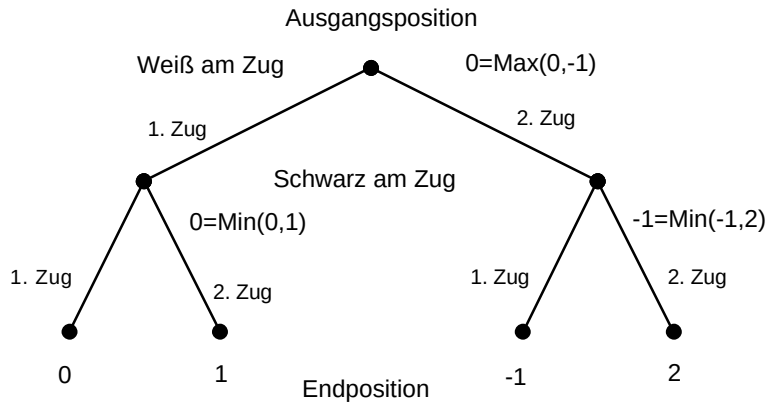



Bild 6.10 Suchbaum des Minimax-Algorithmus

### 6.3.3 Alpha-Beta-Algorithmus

Der Alpha-Beta-Algorithmus ist nicht mehr so einfach wie der Minimax, dafür untersucht er wesentlich weniger Varianten bei gleicher Leistung. Bei genauer Betrachtung der Abbildung des Minimax-Algorithmus fällt auf, daß der äußerst rechte Zug von Schwarz im Suchbaum unnötig ist. Das liegt daran, daß nach der Bewertung von -1 beim ersten Zug von Schwarz (auf den zweiten Zug von Weiß) klar wird, daß Weiß mit seinem zweiten Zug keine bessere Bewertung als 0 (der Wert seines ersten Zuges) erhalten wird, denn Schwarz wird die Bewertung höchstens weiter minimieren. Nach diesem ersten Zug von Schwarz steht also fest: Weiß wird seinen ersten Zug wählen.
Dies bedeutet, daß in einer Rekursionsebene das Minimieren oder Maximieren abgebrochen werden kann, sobald das Maximum bzw. Minimum vom Gegner unter- bzw. überschritten wird (der Abbruch kann auch bei Gleichheit erfolgen).
Der Alpha-Beta Algorithmus implementiert genau diese Vorgehensweise. In dem untenstehenden Pseudocode repräsentiert Alpha den besten Wert bezüglich der Farbe, die gerade am Zug ist (gemäß Color) und Beta den besten Wert der gegnerischen Seite.

```
Konventionen: beim ersten Aufruf enthalten Depth = 1, MaxDepth >= 1 und Color
Werte wie beim Minimax-Algorithmus. Außerdem erhalten Alpha und Beta die
Werte:
       Alpha=MinInteger und Beta=MaxInteger falls Color = White oder
       Alpha=MaxInteger und Beta=MinInteger falls Color = Black

       Prozeduren: wie beim Minimax. Hinzu kommt Return (wie in C).

         Const White      =      1;
               Black      =     -1;
```

```
      MaxInteger =  32767;
      MinInteger = -32768;

Function AlphaBeta (Color, Alpha, Beta,
                           Depth, MaxDepth : Integer) : Integer;
var Value : Integer;

begin
  if Depth = MaxDepth then
     AlphaBeta := EvaluatePosition (Color)

  end else
  begin
     GenerateMoves(Color, MoveList);

     For Each Move in MoveList do
     begin
         MoveForward (Move);

             Value := AlphaBeta (-Color, Beta, Alpha,
                                         Depth +1, MaxDepth);

             if Color = White then
                if Value > Alpha then Alpha := Value;

             if Color = Black then
                if Value < Alpha then Alpha := Value;

         MoveBack (Move);

             if Color = White then
                if Alpha >= Beta then Return Alpha;

             if Color = Black then
                if Alpha <= Beta then Return Alpha;
     end;

     AlphaBeta := Alpha;
  end;
end;
```

Die Analogie zum Minimax ist sehr groß. Im wesentlichen kommen nur zwei Bedingungen hinzu, die unter den oben geschilderten Bedingungen für einen Abbruch der Suche innerhalb einer Rekursionsebene sorgen. Die beiden Schranken Alpha und Beta können anschaulich auch als Suchfenster aufgefasst werden, wobei Alpha den Wert der eigenen bisher besten Variante enthält und Beta den Wert der bisher besten gegnerischen Variante. Das Suchfenster, das zunächst ganz offen ist, wird im Laufe der Suche immer kleiner.

### 6.3.4 Folgerungen für den Praxiseinsatz

Neben diesen einfachen Suchalgorithmen gibt es noch weitere Algorithmen. Diese sind jedoch mehr oder minder speziell auf Schachprobleme zugeschnitten.
Zwei wesentliche Unterschiede zwischen Schach und der Pfadsuche der Roboter müssen bei der Implementierung dieser Algorithmen berücksichtigt werden.

Bei der Berechnung von Zügen (Bewegungsmustern) kann der Roboter von Zug zu Zug unterschiedliche Muster ausführen, auf Schach übertragen bedeutete dies, daß sich die Spielfigur von Zug zu Zug ändert.
Zum anderen sind spieltaktische Finessen, wie das Anbieten von "Figurenopfern" für die Roboter ohne Relevanz, ein Roboter kann nicht einfach verschwinden. Genausowenig können Hindernisse nicht durch "schlagen" aus dem Weg geräumt werden. An diesen Stellen muß die Software geeignete Grenzbedingungen formulieren.
Die Kartendaten liegen in Form eines kartesischen Koordinatensytems bereit. Die globale Intelligenz zerlegt die Gesamtwegstrecke in Teilelemente. Als Kriterium zur Einteilung dient ein prognostiziertes (optimales) Verhalten zur Überwindung dieses Abschnittes. Die globale Intelligenz greift dazu auf eine entsprechende Datenbasis mit vorbereiteten Verhaltensobjekten zurück.
Im Ergebnis dieser Planung entsteht eine Verhaltensobjektliste, die entweder komplett oder zerlegt in Einzelobjekte an die Roboter übergeben wird.
Jedes Verhaltensobjekt gewährt dann den Robotern das notwendige Maß an (Teil-)Autonomie, das zur Absolvierung des Wegelementes ausreichen soll.
Während der Durchquerung der Wegabschnitte verifiziert der Roboter die Karteninformation und den vorgegeben Kurs. Der Zustandsrechner generiert erforderlichenfalls Abbruchbedingungen, die dann zur Berechnung neuer virtueller Zielpunkte führen. Neue Zielpunkte werden vom Zuggenerator vorgeschlagen, die Art der Züge (Bewegungsmuster) hängt dazu von der Prognose der globalen Intelligenz ab. Erweisen sich die Prognosewerte als unbrauchbar, wird auf primitive interne Muster zurückgewichen.
Der Hauptnachteil der internen Vorgaben ist die Kleinräumigkeit der Bewegungen, d.h. es werden nur Bewegungsmuster in der Größe der Reichweite der Sensoren generiert. Da sich der Schätzwert der Abweichung von wahrer und gekoppelter Position von Schritt zu Schritt erhöht, steigt einmal die Gefahr eines finalen Missionsabbruches und zum anderen werden die in der Karte vermerkten Hindernisse bei jeder Iteration vergrößert. Das kann auch dazu führen, daß räumlich enge Passagen zwischen Hindernissen "zuwachsen" und ebenfalls ein Missionsabbruch droht.
Große Probleme bereitet die Steuerung mehrere Roboter, z.B. das Fahren in Formation. Treten Hindernisse auf dem geplanten Formationspfad auf, ist die Entscheidung, wie die Formation diese Problemzonen durchqueren soll, keinesfalls trivial. Interessant wäre hier die Untersuchung, ob mittels Suchbäumen entschieden werden kann, welcher Roboter den größten Erfolg zur Passage von Hindernissen hätte. Wegen der erfolgten Kursmitrechnung können andere Fahrzeuge diesem Weg anschließend folgen.

## 6.4 Erste realisierte Anwendungen

### 6.4.1 Fahren in Formationen

Zu den Herausforderungen im Bereich der mobilen Robotik zählt die kontrollierte Formationsfahrt mehrerer Roboter. Hierzu sind verschiedene Lösungsstrategien denkbar. Die vielleicht häufigste Variante besteht in der kontrollierten Navigation aller Einheiten von einem zentralen Rechner aus. Einer der Vorteile dieses Verfahrens besteht in der ständigen Überwachung und globalen Kontrolle. Speziell beim Roboterfußball scheint dieser Ansatz dem momentanen Stand der Steuerung zu entsprechen.
Was in begrenzten Navigationaräumen, d.h. Spielfeld beim Roboterfußball, vorteilhaft ist, erweist sich bei großräumiger Navigation als problematisch. Es wird zunehmend schwieriger, eine lückenlose Überwachung aller Einheiten zu gewährleisten. Außerdem steigt der für die Kommunikation notwendige Aufwand. Ein weiterer Aspekt ist die nur ungenügende Berücksichtigung der Möglichkeiten der lokal verfügbaren Intelligenz.
Mit Hilfe der bislang gefundenen Algorithmen und technischen Möglichkeiten des Robotersystems "MauSI" wird nun der Versuch einer kontrollierten Formationsfahrt unternommen.
Aus den Simulationsergebnissen von [56] ergibt sich die prinzipielle Möglichkeit aus den Positionsdaten eines Führungsroboters Steuerdaten für "untergeordnete" Robotersysteme abzuleiten.

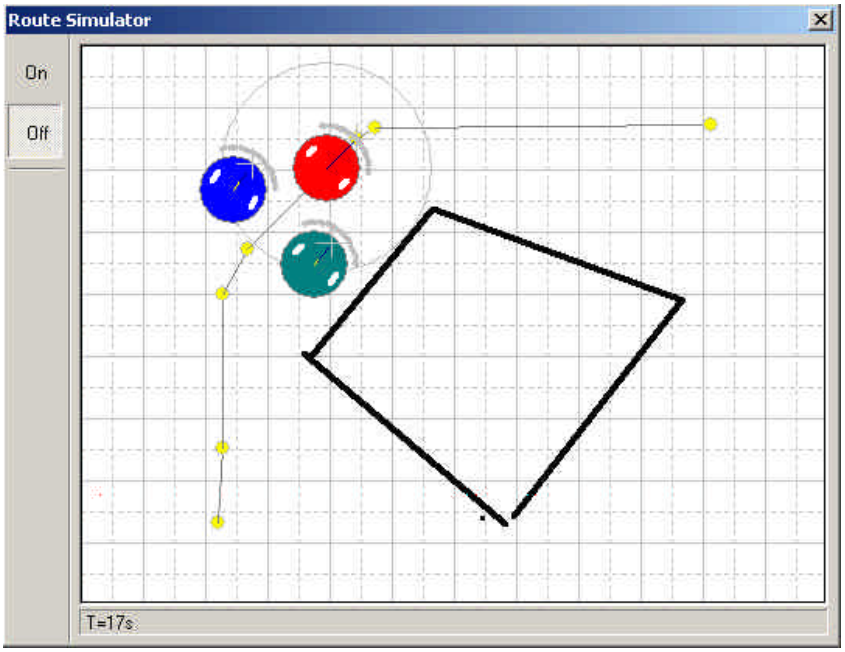



Bild 6.11 Simulation bewegter Roboterformationen [56]

Kennzeichnendes Element dieser Simulation ist die Kursberechnung für nur einen der Roboter der Formation. Die anderen Roboter ermitteln ihrer Kurse selbständig. Dazu benötigen sie Informationen zur Lage ihres Führungsroboters.
In Abwandlung der Simulation wird dem Führungsroboter das Verhalten Zielverfolgung übermittelt und den nachgeordneten Systemen die Art und Weise (Anordnung) der gewünschten Formation bekanntgegeben. Damit liegt die Aufgabenverteilung fest.
Die Realisierung des Versuchsaufbaues benötigt folgende Softwarekomponenten: Koordinatenrechner, Zielpunktrechner und Kommunikationsmodul.

#### 6.4.1.1 Koordinatenrechner

Der Abschnitt 4.2.8. erläutert die laufende Koordinatenmitrechnung in allgemeiner Form. Für die Realisierung als Softwaremodul stellt sich die Forderung nach effizienter Implementierung der Algorithmen unter den Bedingungen des Robotersystems MauSI.
Die Berechnungsformeln sind an und für sich einfache Gleichungen. Damit keine Bewegungselemente verloren gehen, müssen diese Algorithmen mit der Bewegung des Roboters synchronisiert werden, d.h. bei jeder inkrementellen Bewegung (Lichtschranken-Interrupt der Radencoder) muß eine Koordinatennneuberechnung erfolgen.
Bei einer Maximaldrehzahl der Motorwelle von 7000 $U*min^{-1}$ wäre ca. alle 4 ms eine Neuberechnung der Kordinatenwerte nötig. Taktimpulse der Radencoder lösen entsprechende Interrupts im Controller aus. Die Koordinatenneuberechnung beinhaltet die Lösung von Winkelfunktionen (*sin(), cos()*). Eine direkte Berechnung dieser Funktionen in der Interruptroutine würde deren Laufzeit stark verlängern, weitere Interrupts könnten deshalb verloren gehen. Ein Ausweg wäre der Start einer Berechnungstask, die später ausgeführt wird. Da jeder Encoder seperate Tasks starten würde, müßten diese zueinander synchronisiert werden, da sie auf die gleichen Variablen zugreifen (Koordinaten, x, y sowie Drehwinkel).
Eine bessere Lösung stellt die Verwendung einer Lösungstabelle für den Koordinatenoffsett dar. In Abhängigkeit vom Drehwinkel bezogen auf die Lage des Koordinatensystems kann der relative Offset vorher bestimmt werden. Der Drehwinkel des Roboters wird als Index auf die Tabelle interpretiert, die Koordinatenberechnungen sind dann als Additionen bzw. Subtraktionen in der Interruptroutine durchführbar.
Diese Lösung vermeidet zudem die Summierung von Rundungsfehlern bei der Superposition der Rechenergebnisse der beiden Encoder wegen der Verwendung der gleichen Tabelle für beide Berechnungen.

Die Tabelle wird für einen Quadranten berechnet. Eine vollständige Drehung des Roboters um eines seiner Antriebsräder erfordert 4,54 Umdrehungen des anderen Rades. Daraus ergeben sich 1264 Encodertake für eine Drehung. Die Tabelle benötigt deshalb 316 Einträge für die jeweiligen Koordinatenpaare. Da der Controller über genügend internen ROM verfügt, spielt der erforderliche Speicherplatz nur eine untergeordnete Rolle; aus der Sicht des Programmierers wird Rechenzeit gegen Speicher getauscht.

#### 6.4.1.2 Objekterkennung

Für möglichst große Autonomie wird dem Führungsfahrzeug das Verhalten Zielverfolgung aufgeprägt. In ganz einfachen Fällen könnte nun mit Hilfe von Fotosensoren, z.B. Fototransistoren, die Verfolgung eines beleuchteten Objektes "simuliert" werden. Eine solch extreme Abstraktion würde den Leistungen des Robotersystems MauSI nicht gerecht. Die Verwendung eines Bildsensors und die Auswertung dessen Daten, kommt realen Einsatzfällen deutlich näher.
Um die Schwächen der Objekterkennung von monochromen Bilddaten zu vermindern, wird eine Farberkennung implementiert. Der Sensor hat eine Auflösung von 176x144 Bildpunkten, 4:2:2 QCIF-Format. Ein vollständiges Bild benötigt 50688 Byte Speicherplatz. Da der interne Speicher des Controllers zur Aufnahme einer solchen Datenmenge zu klein ist, werden Wege zu Minimierung des Datenvolumens gesucht.
Zwei Vereinfachungen erlauben eine deutliche Reduktion. Bei der Objektdetektion ist nur die x-Koordinate des gesuchten Gegenstandes von Interesse. Damit ein möglichst großer Erfassungsbereich verbleibt, wird deswegen die vertikale Auflösung reduziert. Zur Kompensation des Verlustes vertikaler Bildpunkte wird ein farbiger Stab anstelle des Balles als Zielobjekt verwendet.

Bild 6.12 Objektdetektion bei reduzierter Vertikalauflösung (176 x 36 Pixel)

Die weiße Linie durch das rote Objekt kennzeichnet die x-Koordinate, die der Roboter erkannt hat. Die reduzierte Vertikalauflösung verzerrt die Bildproportionen extrem. Für die eigentliche Erkennung ist dies in diesem Fall ohne Relevanz.
Zur Vergrößerung des horizontalen Erfassungsbereiches dient ein Weitwinkelobjektiv.

Aus der Sicht des Controllers ist die geringere Vertikalauflösung auch hinsichtlich der Rechenzeit von Vorteil. Anstelle von 50688 Byte müssen nur 12672 Byte auf ihre Eigenschaften untersucht werden.

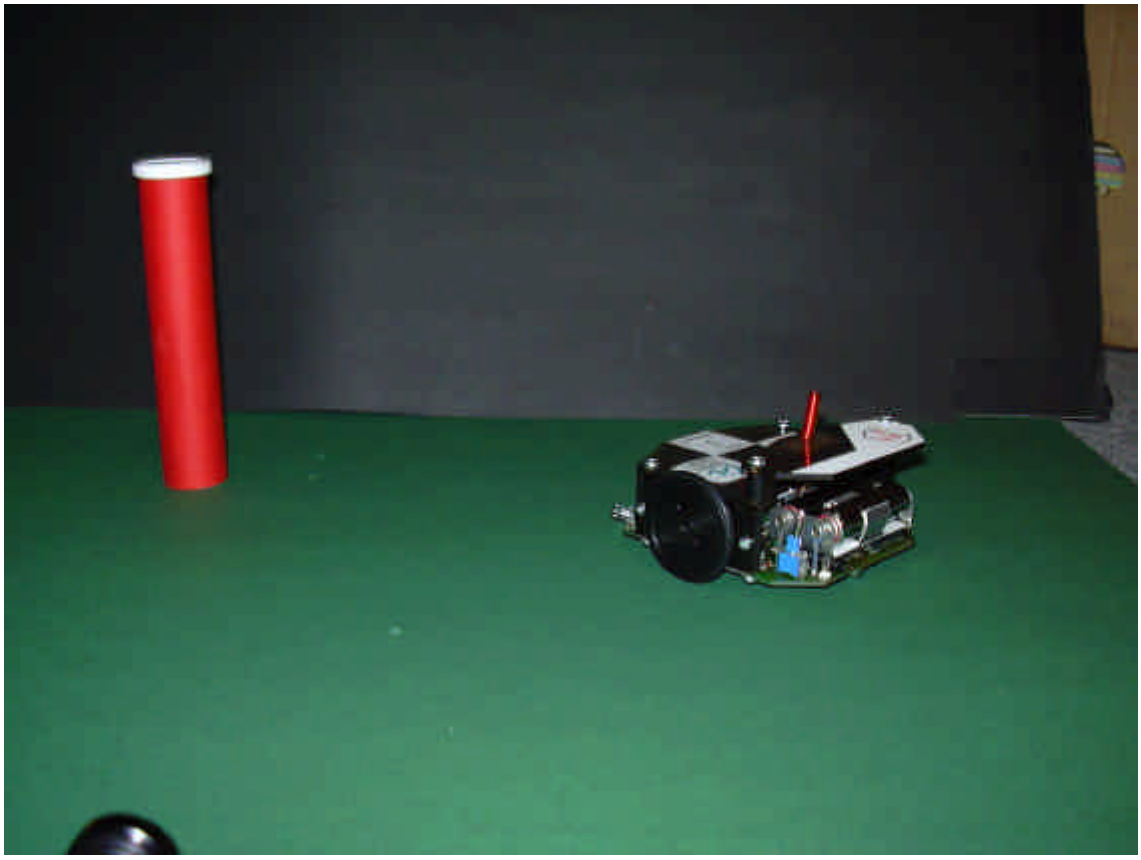


Bild 6.13 Reales Szenario zur Objekterkennung (das Objektiv des suchenden Roboters ist links unten)

Die obenstehende Darstellung zeigt eine wirklichkeitsgetreue Abbildung des Szenarios. Die Objektdetektion in Farbbildern erfolgt mit den gleichen Algorithmen wie bei monochromen Bildfeldern. Zusätzlich werden aus den Intensitätsinformationen der einzelnen Farbwerte Ausschlußkriterien für farblich differierende Objekte gewonnen.
Das Verfahren ist relativ robust, dennoch wird in der Modellumgebung auf kompliziert zu trennende Objekte mit ähnlichen Abmessungen und Farben verzichtet.

#### 6.4.1.3 Formationsfahrt

Mit Hilfe der Informationen zum Zielpunkt und der aktuellen Position können die folgenden Roboter ihre Formation im Verhältnis zum vorausfahrenden Roboter einhalten. Für die Stabilität der Formation ist die Unabhängigkeit zwischen Zieldetektion und Koordinatenberechnung eine wichtige Voraussetzung.

Der Koordinatenrechner erfüllt hier die Aufgabe eines einfachen Zustandsbeobachters. Fehler in der Zieldetektion führen zwar zu falschen Kursen, dürfen aber die Formation nicht stören.

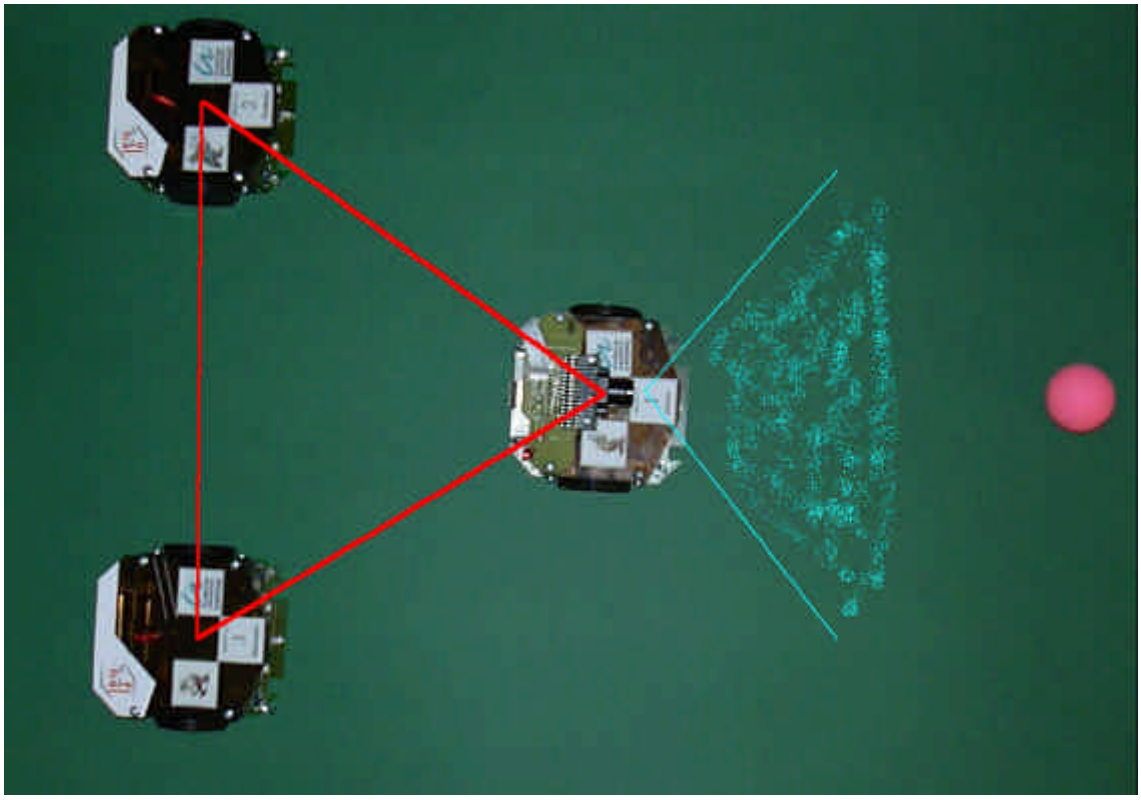


Bild 6.14 Formationsfahrt dreier Roboter

Bild 6.14 verdeutlicht die geometrischen Zusammenhänge. Der Führungsroboter tastet seinen Erfassungsbereich nach dem Zielobjekt ab. Der Bildsensor fungiert dabei als "Fernsensor". Detektierte Objekte werden angefahren, bzw. verfolgt. Werden auch die nachfolgenden Roboter mit Bildsensoren ausgerüstet, entstünden überlappende Suchstreifen. Die versetzte Suchformation sichert die Systeme vor unerkannten Problemen. Schwerwiegende Probleme, z.B. Ausfall des Führungsroboters werden erkannt und können, z.B. an die globale Intelligenz übermittelt werden.
Die Kommunikation der Roboter untereinander erfolgt per Funk. In festen Zeitabschnitten, ca. alle 100 ms, sendet der Führungsroboter eine Statusinformation. Diese beinhaltet seine Koordinaten und den aktuellen Richtungsvektor seiner Bewegung.
Im Rahmen der Genauigkeit der internen Odemetrie kann die Formation aufrecht erhalten werden. Lokal erkennen die Roboter nur Grenzbedingungen, z.B. zu große Annährungen untereinander, mit Hilfe ihrer Infrarotsensoren. Die globale Intelligenz muß die

Kartendaten (Koordinaten) zyklisch aktualisieren. Alternativ hierzu wäre die Gewinnung absoluter Positionsdaten, z.B. per GPS durch die Roboter selbst.

### 6.4.2 Mobile Roboter in der Ausbildung

Neben kommerziellen Auftraggebern aus der Industrie ist die akademische Ausbildung ein beliebtes Einsatzgebiet mobiler Roboter. Die Kombination aus Mechanik, Elektronik und Software gestattet vielfältige Untersuchungen und Experimente in interdisziplinären Facharbeiten.
Außerdem bieten die modellhaften Experimente die Möglichkeit, mit überschaubarem Kostenaufwand gefundene Lösungsansätze auf ihre Praxistauglichkeit zu testen.
Das Robotersystem *MauSI 2* ist robust genug, um als Experimentalplattform in der studentischen Pratikumsausbildung genutzt zu werden. Gleichzeitig sind genügend Ressourcen vorhanden, um das System auch zum Gegenstand weiterer wissenschaftlicher Untersuchungen zu machen. Wichtig ist in diesem Zusammenhang auch die Definition des Roboters als offene Plattform, d.h. viele Teile des Sourcecodes sind frei verfügbar, und es existiert ein leistungsfähiges Entwicklungssystem (C-Compiler, Debugger, etc.) zur Implementation eigener Softwaremodule.
Die Roboterplattform ermöglicht Experimente, die rein auf die lokale Sensorik zurückgreifen, Radencoder und Beschleunigungssensoren zur Zustandsbeobachtung (Koordinatenrechner), IR-Sensoren und Kamerasensor zur Objekterkennung und Hindernisvermeidung, sowie eine leistungsfähige drahtlose Telemetrie zu Ankopplung des Roboters an einen PC.
Die Kombination aus mechanischen Abmaßen, Stromversorgung und Antrieben ermöglicht einen mehrstündigen völlig autonomen Betrieb der Einheit über Batterien oder Akkumulatoren.
Das Robotersystem *MauSI 2* entstand nicht zuletzt auch deshalb, da kein geeignetes kommerziell verfügbares System mit ähnlichem Leistungsumfang im gewählten Kostenrahmen gefunden wurde.

### 6.4.3 Kommerzielle Systeme für Sonderanwendungen

Die Schwierigkeiten in der Definition eines geeigneten "normalverständlichen" MMIs wurden bereits kurz angedeutet. Deutlich entspannter ist die Situation im Handeln der Kombination Spezialist und intelligente Maschine (Roboter). Derartige Roboter sind jedoch bislang nur in Spezialfällen, d.h. in kleinen Stückzahlen, im Einsatz. Beispiele

dafür liegen im Bereich Delaborierung (Bombenentschärfung). Die intelligenten Eigenschaften dieser Systeme sind gemessen an denen, die im akademischen Bereich untersucht werden, bescheiden zu nennen. Es geht fast immer um die Entlastung des Operators von Routinetätigkeiten. Interessant ist in diesem Fall die Frage einer Teilautonomie des Systems, d.h. vom Operator werden Einzelaufgaben vorgegeben, deren detaillierte Ausführung dem Roboter überlassen wird. Eine solche Aufgabe wäre zum Beispiel die Ansteuerung eines bestimmten Geländepunktes. Die Vorgabe des Operators besteht in der Angabe des Zielpunktes und einer ergänzenden Information "schweres rutschiges Gelände".
Der Operator übernimmt hier die Funktion der "globalen Intelligenz". Erreicht der Roboter seinen Zielpunkt ist er für weitere Kommandos bereit, erkennt er auf seinem Weg Hindernisse, deren Überwindung ihm nicht gelingt, fordert er entsprechende Hilfestellung vom Operator an.
Unter Verzicht auf die extrem robusten Mechaniken solcher industriell produzierten Systeme wurde der Versuch unternommen, die Steuertechnik und die gefundene Algorithmik des Systems *MauSI 2* in einen Demonstrator zu integrieren, der über die begrenzten Eigenschaften von *MauSI 2* hinaus, die Möglichkeiten des Einsatzes intelligenter teilautonomer Systeme ausloten sollte.
Die Zielstellung bestand im Aufbau eines ferngelenkten Beobachtungsroboters, der ausgerüstet mit einer Schwenk-Neige-Kamera sowie diverser Sensoren, Aufgaben aus dem Bereich Aufklärung, Bild- und Videodatengewinnung, in von für Menschen schwer zugänglichen Arealen, lösen soll.

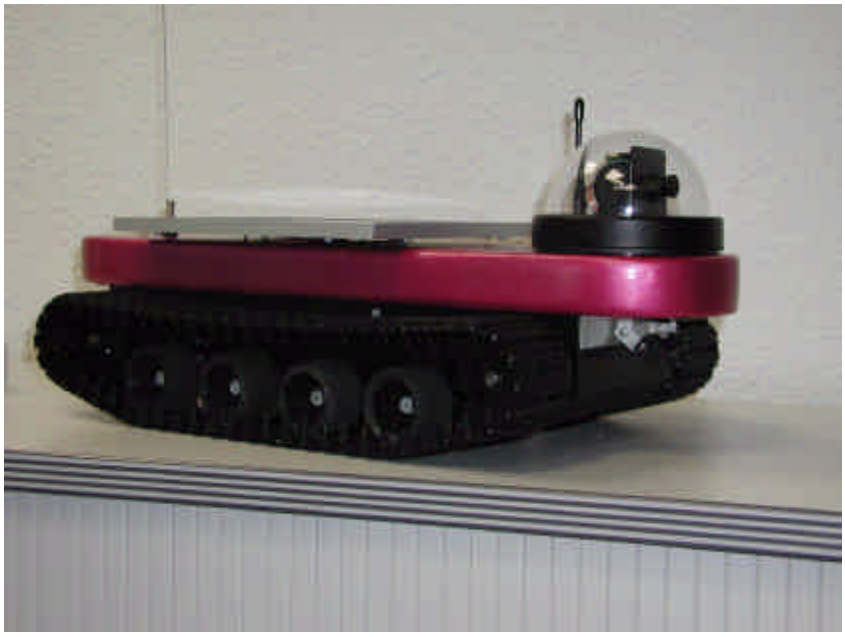


Bild 6.15 Roboter mit Schwenk-Neige-Kamerasystem

Die Abbildung 6.15 zeigt den Prototypen des entwickelten Kamerarobotersystems. Um den Roboter teilautonom fernzulenken, ist eine dazu geeignete Steuersoftware für den PC entstanden. Die Bediensoftware eröffnet dem Operator den Zugang zu den diversen Steuerfunktionen des Roboters.
Ein Bildschirmausdruck dieses Programms ist in Abbildung 6.16 zu sehen. Der Roboter hat die Aufgabe mit der eingebauten Schwenk-Neige-Kamera Videodaten an den Steuer-PC zu übertragen. Das Videobild ist zentraler Bestandteil der Bediensoftware. Da aber der Roboter kein "einfaches nur ferngelenktes" System ist, braucht der Operator Zugriff auf die internen Daten des abgesetzt arbeitenden Systems.
Zur Darstellung von Batteriezustand bzw. zur Erkennung weiterer interner Sensorwerte, wie z.B. Lage des Roboters sind eine Reihe grafischer Anzeigen im rechten Teil der Bedienoberfläche plaziert (zum Teil vom "Robot Way Tracer" Window verdeckt).

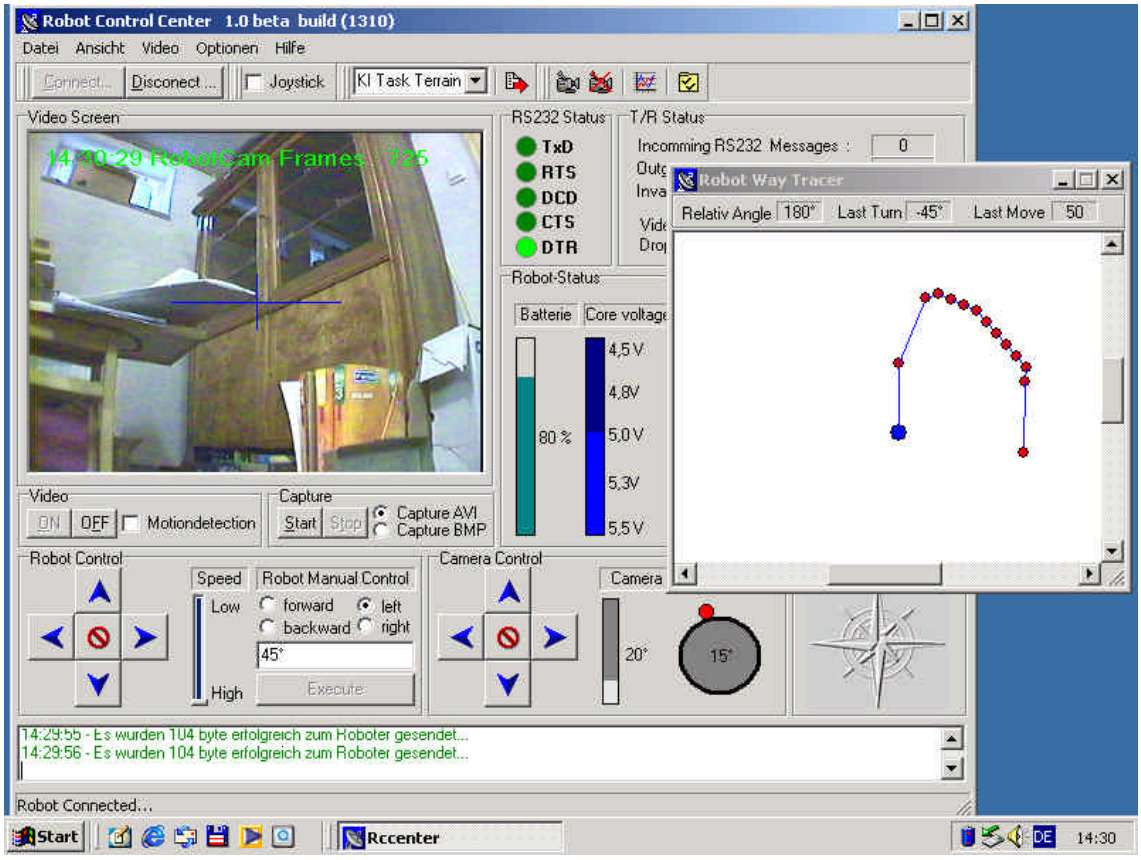


Bild 6.16 Bildschirmansicht der Bediensoftware für den Kameraroboter

Sehr wichtig für die teilautonome Steuerung des Roboters ist die Protokollierung des abgefahrenen Weges. Die Daten des "Robot Way Tracer" stehen z.B. als Wegmarkierung für eine selbstständige Rückkehr des Fahrzeuges zum Ausgangspunkt zur Verfügung.

# 7 Zusammenfassung und Ausblick

Eine Grundforderung aller Untersuchungen und Experimente war die Kosten-Nutzen-Relation. Die Frage der Kosten ist dabei vergleichweise einfach zu beantworten. Der Nutzenaspekt ist dafür erheblich schwieriger einzuschätzen.
In den einführenden Abschnitten sind Roboter bzw. roboterähnliche Konstruktionen genannt worden, die teilweise bereits heute im Einsatz sind. Weitere Entwicklungen sind soweit gediehen, daß sie in Kürze zum Einsatz gelangen könnten.
Ein wichtiger Punkt in der Nutzens-Relation ist die Aktzeptanz von Robotern in vorwiegend "menschlichen" Umwelten. In einem Vortrag an der TU Ilmenau [37] wurde dieser Aspekt des Einsatzes angesprochen. Beim Gegenstand der Untersuchung handelt es sich um ein Assistenzsystem, daß z.B. Hilfestellung bzw. ergänzende Informationen zum Sortiment eines Baumarktes bereitstellt. Die Verknüpfung der Datenbestände (Angebot, Preise, Lokalisation der gewünschten Produkte) mit den Möglichkeiten und Erfordernissen eines autonomen Systems (Selbstlokalisation, Aufbau eines Kontaktes zwischen Mensch (Kunde) und Maschine) steht noch am Anfang der Entwicklung. Insbesondere die "unscharfe" Kommunikation zwischen Menschen, sei es z.B. die diversen Bestätigungen einer Frage: Nicken mit dem Kopf, zustimmende Handbewegung, akustische Antwort in verschiedenen Dialekten, stellen enorme Anforderungen an ein allgemeinverständliches Mensch-Maschine-Interface (MMI).

## 7.1 Entwicklungstendenzen

In einem eher allgemein philosophisch formulierten Buch über Robotik [39] stellt der Autor die Behauptung auf, daß die Robotik eine mehr oder minder "kannibalistische" Wissenschaft darstellt. Es wird an dieser Stelle nicht versucht, über die Formulierung dieser Aussage zu diskutieren.
Richtig ist jedoch sicher, daß sich die Robotik soweit wie möglich der Entwicklungen anderer Gebiete der Technik bedient. Dies geschieht zum Teil aus der Zwangslage der geringen produzierten Stückzahlen von Robotern heraus. Eine Anzahl von Technologieprodukten ist uns so sehr vertraut und gemessen am Funktionsumfang zum Teil geradezu lächerlich billig, so daß kaum jemand einen Gedanken an die zugrunde liegende Hochtechnologie verschwendet. Als Beispiel sei hier auf die Entwicklung von Mobiltelefonen verwiesen.

Preiswert werden technische Produkte nur durch massenhafte Produktionsstückzahlen. Insofern darf man die Robotik sicherlich als kannibalistisch bezeichen, da sie sich meist ungehemmt mit Geräteteilen der Massenproduktion schmückt.

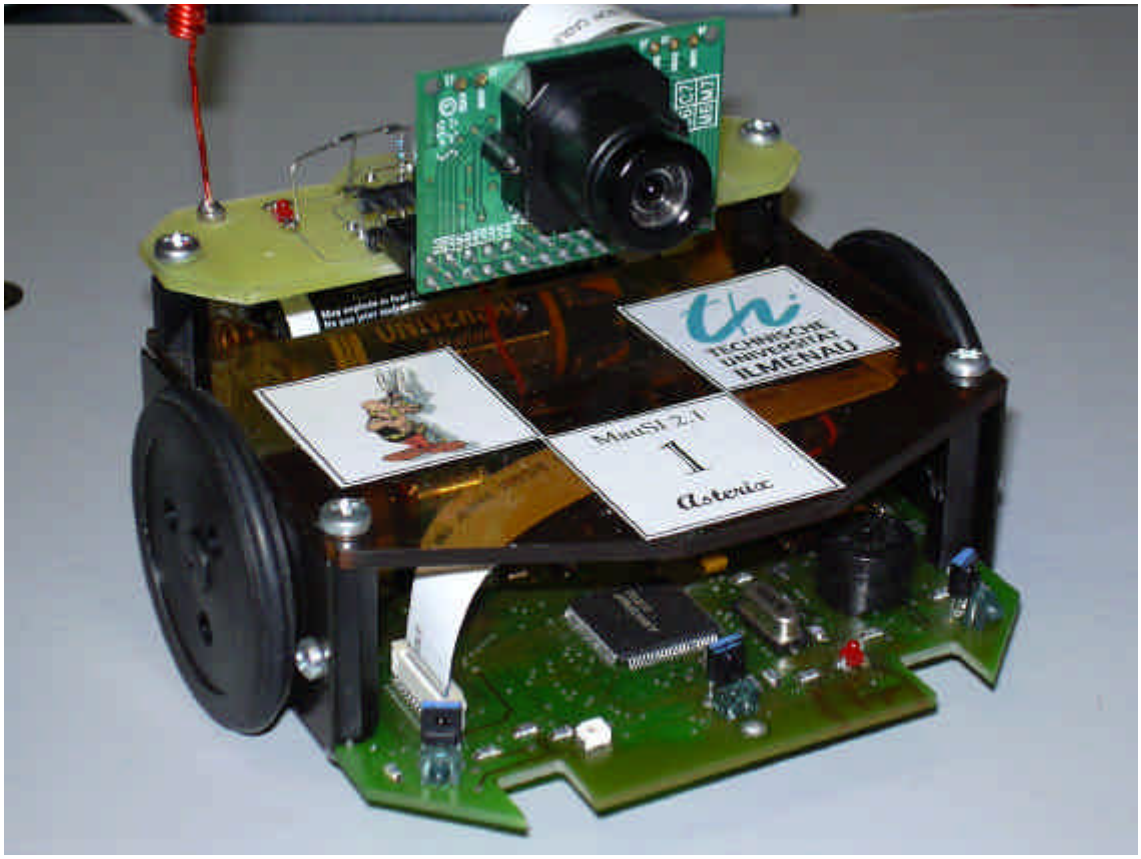



Bild 7.1 Robotersystem MauSI 3 (MauSI 2 mit Kamera)

Im Verlaufe vieler Experimente mit den Robotern "MauSI 1" und "MauSI 2" spielten die begrenzenden Parameter von Sensorik und Telemetrie immer eine entwicklungshemmende Rolle.

Der Datenaustausch zwischen den Robotersystemen erfolgte anfangs mit 1200 Baud. Diese geringe Datenrate zwang zum Teil zu nicht hinnehmbaren Einschränkungen hinsichtlich der Funktionalität. Erst die Verfügbarkeit kommerzieller *low power devices* im 433MHz ISM löste das Problem der Datenrate. Mit zunächst 9600 Baud und später 38400 Baud konnte die Koordination von Roboter zu Roboter bzw. von Roboter zum PC verbessert werden.

Ähnlich zeigte sich die Situation auf dem Gebiet der Abstandsdetektoren bzw. bei der Hinderniserkennung. In vielen kommerziellen Einsatzfällen liefern Laserscanner exakte Entfernungsmessungen. Solche Sensoren sind jedoch für die Plattform "MauSI 1" oder "MauSI 2" ungeeignet.

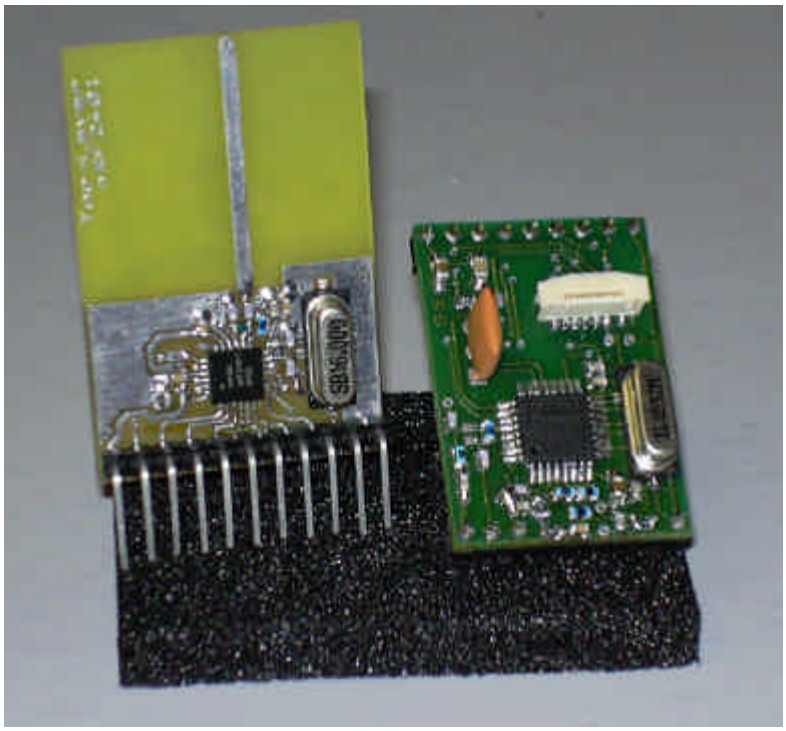


Bild 7.2 High-Speed Datenmodem für das 2,4 GHz bzw. 433 MHz ISM Band

Auch hier ermöglichte der starke Preisverfall bei CMOS-Kamerasensoren eine Alternative. Die Leistung des 16-Bit Steuercontrollers ermöglicht eine Bilddatengewinnung auf lokaler Ebene. Der Roboter wurde "sehend", leider nur mit Schwarz-Weiß-Bildern.
Die Übertragung von Farbaufnahmen scheiterte zuerst am Flaschenhals Datenrate der eingesetzten Funkmodule.
In der Abbildung 7.2 sind neuentwickelte High-Speed-Funkmodule zu sehen, die für den Einsatz der Roboterplattform "MauSI 3" konzipiert wurden. Das rechte Modul arbeitet im 433 MHz-ISM-Band und erreicht 76,8 kBaud unkomprimiert. Ein eingebauter separater Mikrocontroller mit ADPCM-Datenkompression erhöht die Datenrate auf 150 kBaud. Der Baustein ist als direktes Replacement für das bislang verwendete Funkmodul konzipiert.
Eine noch größere Datenrate von 1 MBaud ermöglicht das linke Modul. Es arbeitet im 2.4 GHz ISM-Band und erlaubt eine Bewegtbildübertragung vom Roboter aus.
Letztendlich läßt sich feststellen, daß die eingangs formulierten Forderungen nach einem leistungs- und ausbaufähigem Robotersystem in einem engen Kostenrahmen weitgehend realisiert worden sind.
Das System verfügt über Sensoren die eine Nahbereichserfassung zur Kollisionsvermeidung ermöglichen, ist genügend mobil um teilautonom Aufgaben zu übernehmen und zu lösen, und besitzt entsprechende Schnittstellen zur Erweiterung des Funktionsumfanges.

Wichtig ist auch eine gewisse Robustheit um den Anforderungen eines Praktikumseinsatzes zu genügen.
Der überwiegende Teil der Steuersoftware ist in C geschrieben. Dies sichert eine gute Wartbarkeit des Codes und ist im Bereich des Embedded Control als *de facto* Standard anzusehen.
Besonders nützlich sind Schnittstellen die es ermöglichen, komplexe Regeln oder Algorithmen auf dem PC mit Standardprogrammen wie MathLab/Simulink theoretisch zu untersuchen und dort gewonnene Erkenntnisse mit Hilfe eines Code-Generators als ANSI-C Funktionen auf dem Roboter anschließend praktisch zu erproben.
Auch das "Killerkriterium" Kosten, ist halbwegs gebannt. Die Erprobung des Kameraroboters mit anschließender Präsentation des Systems vor ausgewähltem Publikum bestätigt die Richtung des Entwicklungsweges.

*Vollkommenheit entsteht offensichtlich nicht dann,*
*wenn man nichts mehr hinzuzufügen hat,*
*sondern wenn man nichts mehr wegnehmen kann.*
*Die Maschine in ihrer höchsten Vollendung wird unauffällig.*

*Antoine de Saint-Exupéry*

# 8 Literaturverzeichnis


[1] J. Jones, A. Flynn: "Mobile Roboter - Von der Idee zur Implementierung", Addison-Wesley, 1996

[2] D. Kortenkamp,R.P. Bonasso, R Murphy: "Artifical Intelligence and Mobile Robots", Menlo Park, California

[3] P. Kampmann: "Ein topologisch strukturiertes Weltmodell als Kern eines Verfahrens zur Lösung von Navigationsaufgaben bei mobilen Robotern", VDI-Verlag 1992

[4] P. Hoppen: "Autonome mobile Roboter, Echtzeitnavigation in bekannter und unbekannter Umgebung", BI-Wissenschaftsverlag 1992

[5] S.Y. Harmon: "Autonomous Vehicles", in "Encyclopedia of Artifical Intelligence", John Wiley&Sons, 1987, S. 39-45

[6] S.Y. Harmon: "Mobile Robots", in "Encyclopedia of Artifical Intelligence", John Wiley&Sons, 1987, S. 39-45

[7] Wernstedt, Eichhorn: "Softwaremodell zur Modellierung der Steuerparameter für Unterwasserfahrzeuge", Facharbeit TU Ilmenau

[8] U. Nehmzow: "Mobile Robotics: A Practical Introduction", Springer-Verlag Berlin Heidelberg New York London 2000

[9] Fahrerlose Transportsysteme, Firmenschrift Fa. S-Elektronik www.s-elektronik.de

[10] I. P. Pawlow: "Der Vater der Verhaltenpsychologie" Multimediapräsentation www.pcz.uni-dortmund.de/projekte/multimediagestalten/dokument/kap2/klassi.htm

[11] Walter, W. Grey: "The Living Brain" (1953) New York: W. W. Norton

[12] Wiener, Norbert: "Cybernetics; or, Control and Communication in the Animal and the Machine" New York: Wilney

[13] Ch. Hülm, S. Pietzsch: "Vom Kerbholz zur Rechenanlage" Kinderbuchverlag Berlin 1977

[14] M.Edelhart, D.Garr:"Das Computer-Lesebuch" Heyne Verlag München 1984

[15] Kurz, G. (Hrsg.): "Analoge Schaltungen" 1. Auflage, Militärverlag der DDR, Berlin 1979

[16] Hoffmeier, Konrad: "Ladetechniken für beschleunigtes Laden von Nickel-Kadmium-Akkumulatoren, Studienjahresarbeit"; TU Ilmenau - Fakultät für Informatik und Automatisierung

[17] Uhlitzsch, Elmar: "Implementierung von Infrarot-Sensoren zur Lage- und Abstandserkennung von mobilen Robotern", Studienjahresarbeit; TU Ilmenau - Fakultät für Informatik und Automatisierung

[18] Weinberg, Harvey: "Accelerometers - Fantasy and Reality", Analog Dialoge, Volume 32, 2001, S. 63 (Firmenschrift der Fa. Analog Devices)

[19] Altenburg, J.; Altenburg, U. : "Mobile Roboter - Vom einfachen Experiment zur künstlichen Intelligenz"; Carl Hanser Verlag, München 1999

[20] Altenburg, J; Altenburg, U.: "Modell eines autonomen Systems mit variabler Intelligenz", 45. Internationales wissenschaftliches Kolloqium, Ilmenau 2000 (Tagungsband)

[21] Man, R.; Willrich, C.: "A Minimalist Multitasking Executive", Circuit Cellar #101 1998

[22] Schlüter, Gerd.: "Digitale Regelungstechnik - Grundlagen zeitdiskreter Systeme", Carl Hanser Verlag, München 2000

[23] A. Rowe, C. Rosenberg, I. Nourbakhsh "A Simple Low Cost Color Vision System", www-2.cs.cmu.edu/~cmucam/

[24] Pratt, William K.: "Digital Image Processing", A Wiley-Intersience publication, New York 1991

[25] Müller, Heinrich : "Vorlesung Digitale Bildverarbeitung" Universität Dortmund, Informatik VII, http://ls7-www.cs.uni-dortmunt.de

[26] Eckhard, Uwe : "Studienjahresarbeit zur Ermittlung und Berechung von Bahnparametern zur Steuereung eines autonomen Roboters", TU Ilmenau - Fakultät für Informatik und Automatisierung

[27] Richey, Rodger: "Adaptive Differential Pulse Code Modulation using PICmicro™ Microcontrollers, Application Note AN643, www.microchip.com

[28] Rupp, Torsten: "Transportroboter James", Forschungszentrum Informatik Karlsruhe, Bereich MMR;
http://www.fzi.de/mmr/german/projects/robot/details/gallery.html

[29] Dickmanns, Ernst: "Fahrzeuge lernen sehen", Fakultät für Luft- und Raumfahrttechnik, Universität der Bundeswehr München

[30] Buehler, Martin: "RHex - Centre for Intelligent Machines", McGill University www.cim.mcgill.ca/~buehler

[31] Hogg, Robert: "Creepy Crawlers May Unreval Web of Planetary Mysteries", Jet Propulsion Laboratory, Pasadena www.jpl.nasa.gov/releases/2002/232.cfm

[32] Kampmann, Peter: "Ein topologisch strukturiertes Weltmodell als Kern eines Verfahrens zur Lösung von Navigationsaufgaben bei mobilen Robotern" VDI-Verlag, Düsseldorf 1992

[33] Werner, Christian: "Bewegungsplanung der Spieler im RoboCup", Hauptseminar RoboCup, Universität Stuttgard, Fakultät Informatik, Sommersemenster 2002

[34] Lochmann, Dietmar: "Digitale Nachrichtentechnik", 2. Auflage Verlag Technik Berlin 1997

[35] Schmidt, Peter: "Hard- und softwaretechnische Realisierung einer Steuerung für einen mobilen Roboter mittels Fuzzy-Logik", Diplomarbeit TU Ilmenau 1995 Inventarisierungsnummer 200-95D-088

[36] Glotzbach, Thomas: "MathLab/Simulink-Simulation MauSI in virtueller Welt" unveröffentlichte Facharbeit, TU Ilmenau 2002

[37] H.-J. Boehme: "Serviceroboter und intuitive Mensch-Roboter-Interaktion" Schriftenreihe des FG Neuroinformatik der TU Ilmenau, Report 1/02 , ISSN 0945-7518

[38] Gero v. Randow: "Roboter - Unsere nächsten Verwandten" Rowohlt Verlag Hamburg 1997

[39] Heuner, Andre: "Website JavaChess" http://www.trinimon.de/Chess/Chess Game/Develop/chs_min.htm

[40] www.menzelphoto.com/gallery/big/robo1.htm

[41] Begleitheft "Mobile Robots II" der Fa. Fischertechnik zum Baukastensystem "Robotik"

[42] Wetterling, Friedrich: "Positionsbestimmung mit einem Beschleunigungssen-sor" Studienarbeit TU Ilmenau, Fakultät Informatik und Automatisierung 2003

[43] Bauer, Friedrich: "Entzifferte Geheimnisse - Codes und Chiffren", Springer Verlag 1995

[44] Der Spiegel: "Nie wieder aufräumen, Eine neue Generation von Spielzeugen..", Der Spiegel 7/1998, Seite 164 -165

[45] Webb, Barbara: "Eine elektromechanische Grille", Spectrum der Wissenschaft 5/97, Seite 78 ff.

[46] Hesselmann, N.: "Digitale Signalverarbeitung", Vogel Buchverlag 1987

[47] Philippow, E. (Hrg.): "Taschenbuch Elektrotechnik", Verlag Technik Berlin 1987

[48] Ewe, T.: "Der Robot-Schwarm", Bild der Wissenschaft 6/1998 Seite 46 ff.

[49] Reuber, C.: "Ballon mit Ultraschallortung aus Berlin", ELEKTRONIK 1/1996, Seite 18

[50] Cruse, Dean, Ritter: "Prärationale Intelligenz", Spektrum der Wissenschaft; Dossier: Roboter.. 4/98

[51] Baum, Dave: "Lego-Mindstorm", Gallileo-Verlag 2000

[52] Beierlein, Hagenbruch: "Taschenbuch Mikroprozessortechnik", Fachbuchverlag Leipzig 2000

[53] Labrosse, Jean L. :"µC/OS - The Real-Time Kernel", R & D Publikations

[54] Altenburg, J.: "SOPHOCLES - A solar powered MSP430 Robot", Circuit Cellar #147, 2002

[55] Wernstedt, Glotzbach, Altenburg: "Methoden zur Detektion und Vermeidung von Hindernissen in der mobilen Robotik", Symposion Unmanned Ground Vehicle, Mannheim 2002

[56] Altenburg, Uwe: "Strategie zur koordinierten Steuerung von autonomen mobilen Multirobotersystemen", Facharbeit, TU Ilmenau 2003

# Abbildungsverzeichnis

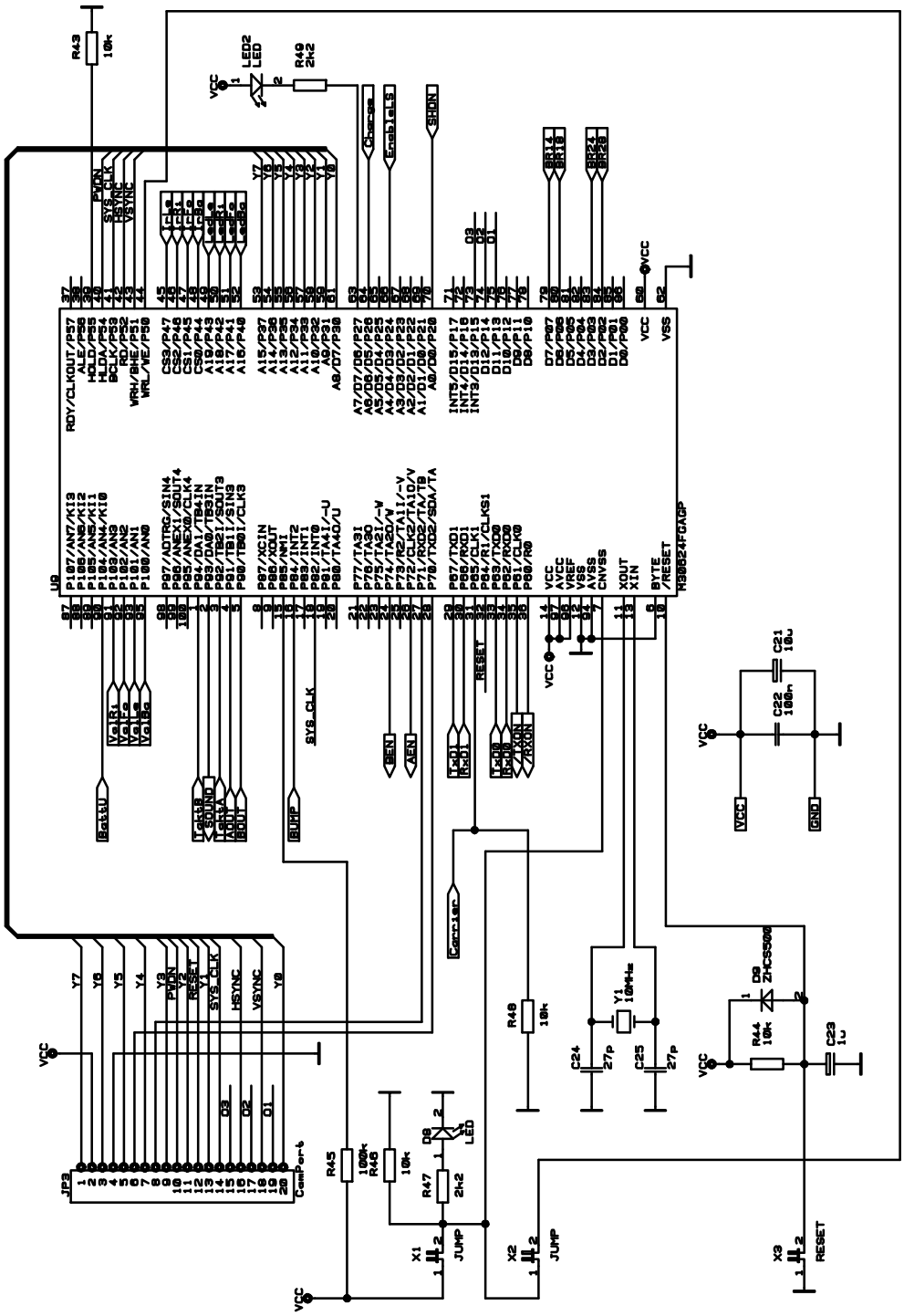
# Anhang

## Betriebsdaten des DC-Motors FTB 20-08

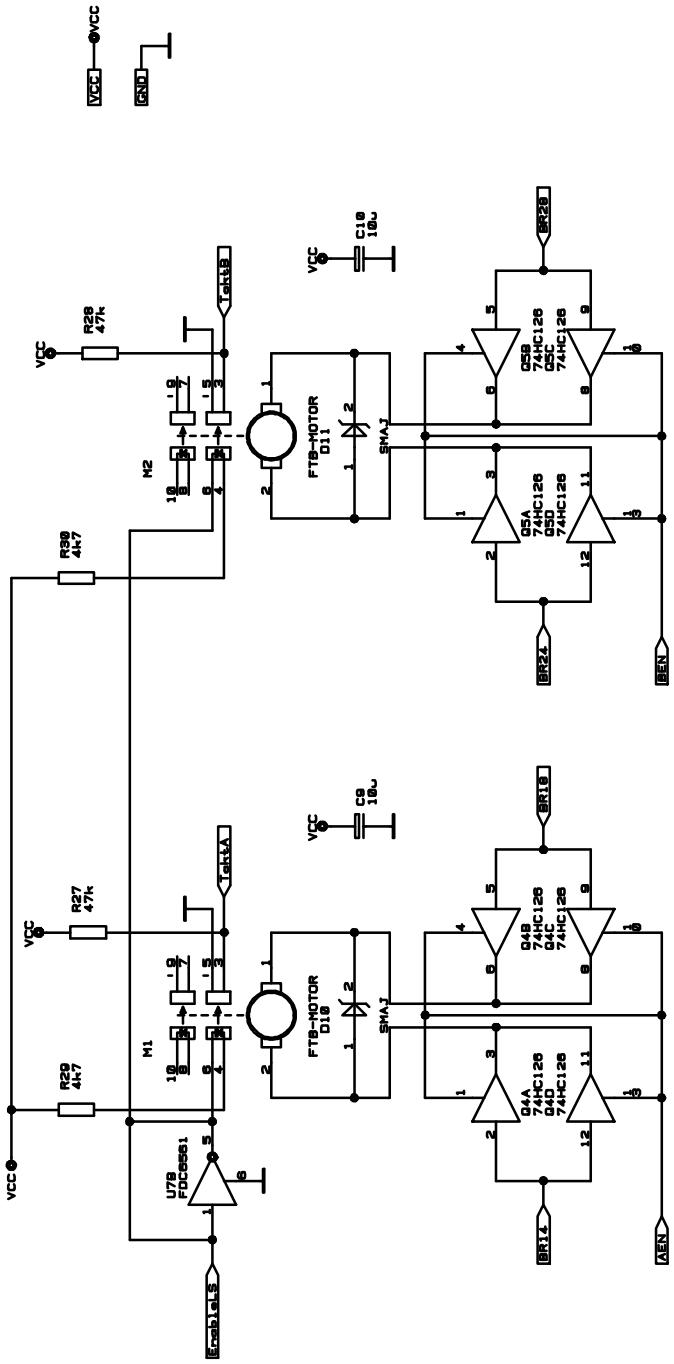
**FTB 20-08**

| Motortyp | | 20-08 | | -03.00 | -06.00 | -12.00 | |
|---|---|---|---|---|---|---|---|
| 1 | Nennspannung | $U_N$ | | 3 | 6 | 12 | V |
| 2 | Anschlusswiderstand | R | | 7,7 | 28 | 114 | Ω |
| 3 | Abgabeleistung, max. bei $U_N$ | $P_{2\ max.}$ | | 0,263 | 0,284 | 0,281 | W |
| 4 | Leerlaufdrehzahl | $n_o$ | | 7.000 | 7.000 | 6.400 | rpm |
| 5 | Leerlaufstrom | $I_o$ | | 0,020 | 0,013 | 0,006 | A |
| 6 | Anhaltemoment | $M_H$ | | 1,513 | 1,650 | 1,778 | mNm |
| 7 | Reibungsdrehmoment | $M_R$ | | 0,08 | 0,09 | 0,10 | mNm |
| 8 | Drehzahlkonstante | $k_n$ | | 2.460 | 1.234 | 564 | rpm/V |
| 9 | Generator-Spannungskonstante | $k_E$ | | 0,41 | 0,81 | 1,77 | mV/rpm |
| 10 | Drehmomentkonstante | $k_M$ | | 3,88 | 7,69 | 16,89 | mNm/A |
| 11 | Stromkonstante | ki | | 0,26 | 0,13 | 0,06 | A/mNm |
| 12 | Steigung der n-M-Kennlinie | $\Delta n/\Delta M$ | | 4.626 | 4.242 | 3.599 | rpm/mNm |
| 13 | Anschlussinduktivität | L | | 0,23 | 1,4 | 4,7 | mH |
| 14 | Mech. Anlaufzeitkonstante | $\tau_m$ | | 25 | 25 | 25 | ms |
| 15 | Rotorträgheitsmoment | J | | 0,55 | 0,55 | 0,55 | gcm² |
| 16 | Betriebstemperaturbereich: | | | | | | |
| | – Motor | | –20… +65 | | | | °C |
| | – Rotor, max. zulässig | | –20… +80 | | | | °C |
| 17 | Rotorwellenlagerung | | Kunststoff/Messing (Standard) | Kunststoff/Sinterbronze (optional) | | | |
| 18 | Magnetmaterial | | NdFeB | | | | |
| 19 | Gewicht | | 10 | | | | g |
| **Empfohlene Werte** | | | | | | | |
| 20 | Drehzahl bis | $n_{e\ max.}$ | | 6.000 | 6.000 | 6.000 | rpm |
| 21 | Dauerdrehmoment bis | $M_{e\ max.}$ | | 0,4 | 0,4 | 0,4 | mNm |
| 22 | Thermisch zulässiger Dauerstrom | $I_{e\ max.}$ | | 100 | 50 | 30 | mA |

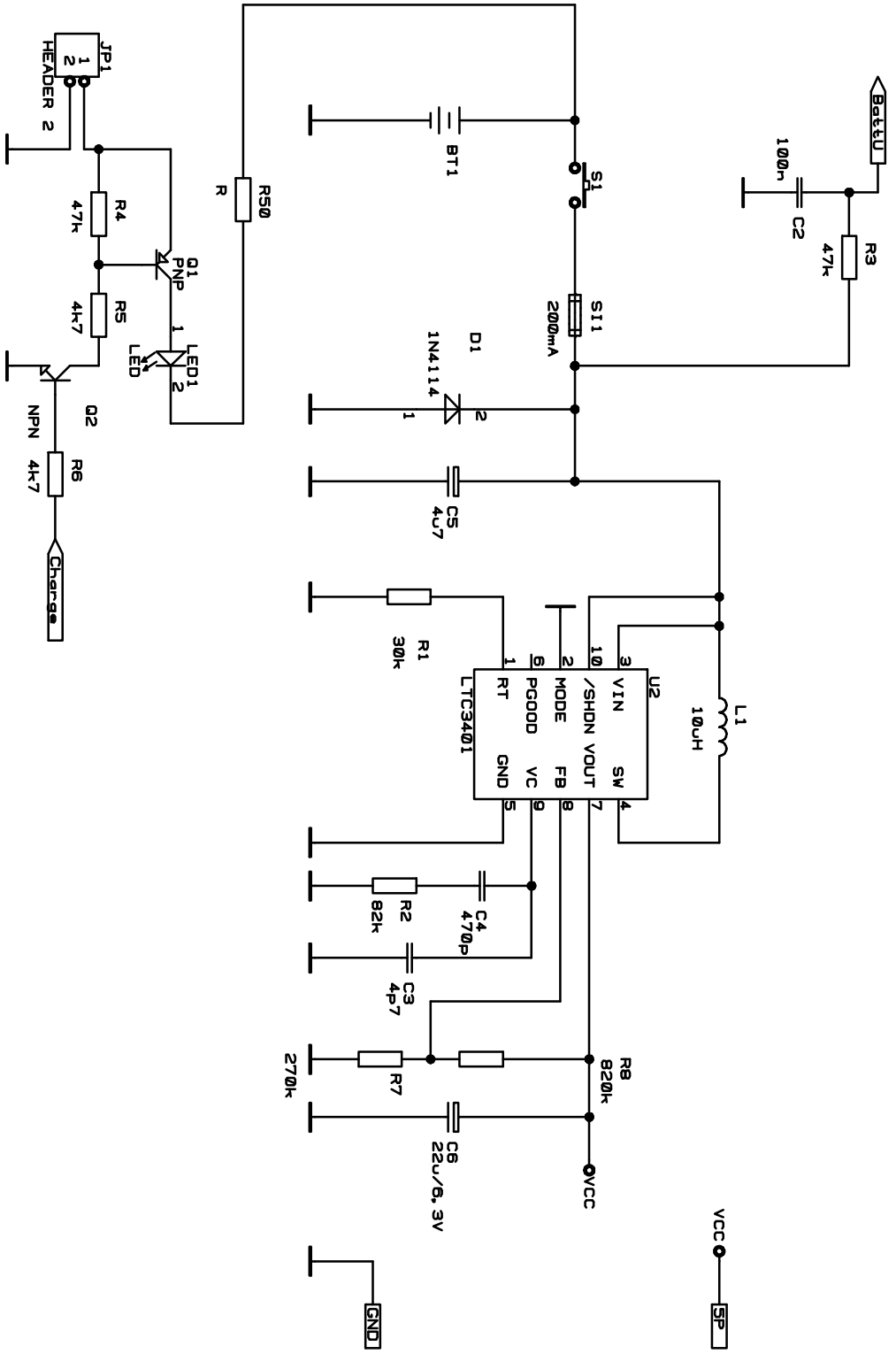
CPU-Einheit



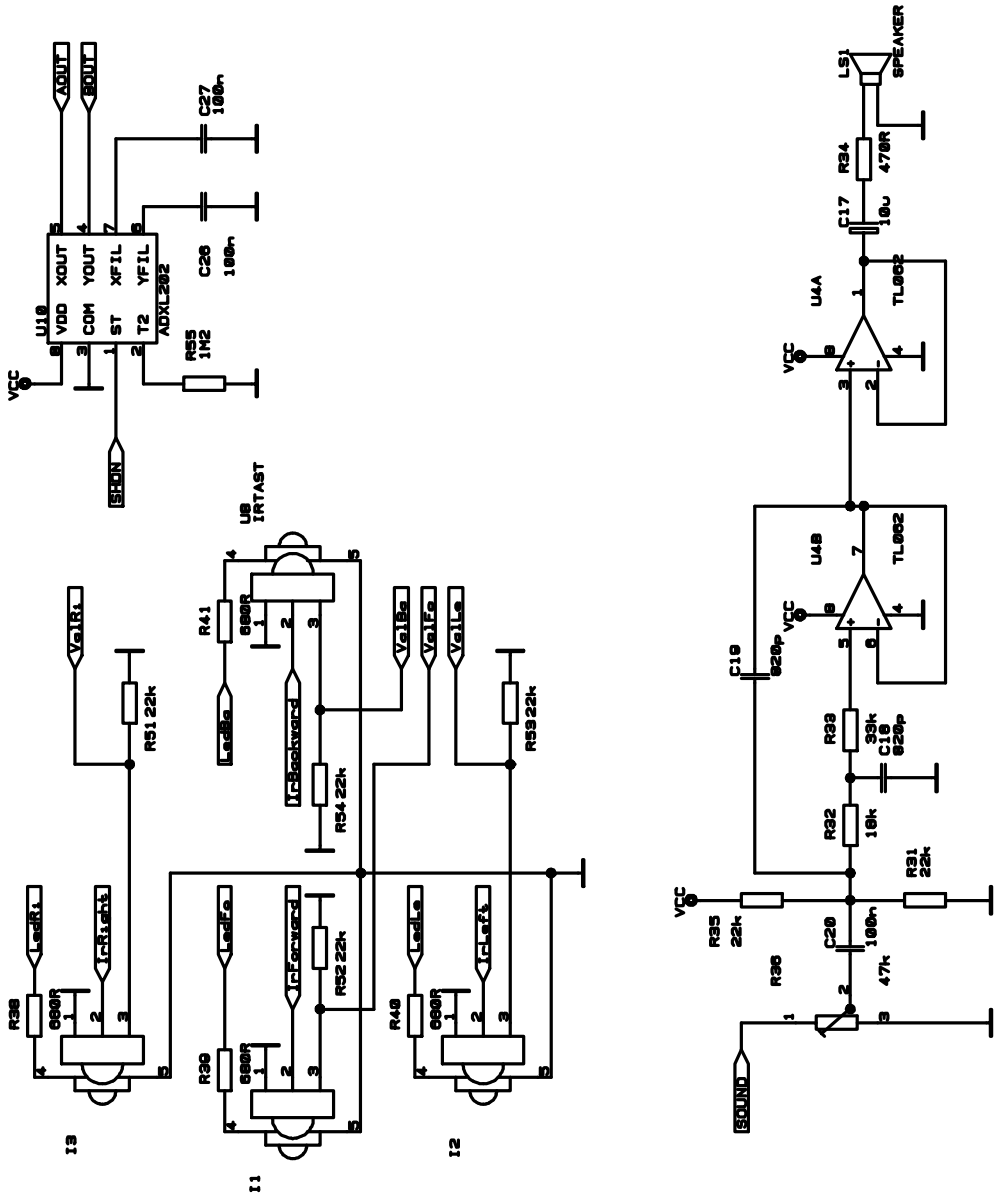
Motoransteuerung



Stromversorgung



Infrarot-Sensoren



CMOS-Kamerasensor